UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Diseño e implementación de un sistema para la monitorización de protocolos IoT

Máster Universitario en Ingeniería de Telecomunicación

Autor: David Sanches Gómez
Tutor: Jorge Enrique López de Vergara Méndez
Departamento de Tecnología Electrónica y de las Comunicaciones

FECHA: Febrero 2021

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA PARA LA MONITORIZACIÓN DE PROTOCOLOS IOT

Autor: David Sanches Gómez
Tutor: Jorge Enrique López de Vergara Méndez

High Performance Computing and Networking Research Group
Dpto. de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Febrero 2021

Resumen

El desarrollo en el campo de las telecomunicaciones conlleva el deseo de incluir sistemas que permitan el intercambio de información entre diferentes dispositivos que se encuentran desplegados a día de hoy de forma ubicua. Por esta razón, surge el IoT, Internet de las Cosas (*Internet of Things*), ante la necesidad de dotar de tecnología a todo tipo de instrumentos de manera que se pueda generar una transmisión de datos entre ellas.

El IoT supone la interconexión de cualquier dispositivo a través de la red. Esta comunicación se ha de realizar mediante el uso de protocolos específicos dada la naturaleza de los elementos de una red IoT, ya que pueden ser desde un simple sensor a una maquina compleja. Dos de los protocolos más importantes, y los tratados en el presente documento, son el Protocolo de Aplicación Limitado (*Constrained Application Protocol*, CoAP) y el protocolo de Transporte de Telemetría por Cola de Mensajes (*Message Queue Telemetry Transport*, MQTT).

Debido a las peculiaridades de estos protocolos, en este Trabajo Fin de Máster se desarrolla un sistema de monitorización que permita conocer el estado de salud de una red IoT, así como poder detectar posibles anomalías dentro de ella. Para este cometido, se analiza el tráfico MQTT o CoAP capturado, generando unos archivos de registros, los cuales serán volcados en unas bases de datos, de manera que se pueda acceder a ellas para así poder representar gráficamente los resultados obtenidos. El objetivo de esta representación es permitir a un gestor de red conocer el estado de la red IoT analizada de una forma clara e intuitiva, a través de un cuadro de mando.

También, se han realizado diferentes verificaciones con las que se ha podido validar el correcto funcionamiento del sistema de monitorización en su conjunto.

Palabras clave

Redes IoT, MQTT, CoAP, Monitorización, Gestión de Red, Cuadro de Mando.

Abstract

The development in the field of telecommunications entails the desire to include systems that allow the exchange of information between different devices that are deployed ubiquitously today. For this reason, the IoT, Internet of Things, arises from the need to provide technology to all sorts of instruments so that data transmission between them can be generated.

The IoT involves the interconnection of any device through the network. This communication has to be done through the use of specific protocols given the nature of the elements of an IoT network, since they can range from a simple sensor to a complex machine. Two of the most important protocols, and the ones discussed in this document, are the Constrained Application Protocol (CoAP) and the Message Queue Telemetry Transport protocol (MQTT) .

Due to the peculiarities of these protocols, this Master Thesis develops a monitoring system that reveals the health status of an IoT network, as well as to detect possible anomalies within it. For this purpose, the captured MQTT or CoAP traffic is analyzed, generating log files, which will be stored in databases, so that they can be accessed in order to graphically represent the results obtained. The purpose of this representation is to allow a network manager to know the status of the IoT network analyzed in a clear and intuitive way, through a dashboard.

Also, different verifications have been carried out to validate the correct operation of the monitoring system as a whole.

Keywords

IoT Networks, MQTT, CoAP, Monitoring, Network Management, Dashboard.

Agradecimientos

Lo primero comenzar agradeciendo a mi tutor, Jorge E. López de Vergara, por de nuevo permitirme realizar el proyecto con él de tutor. Después de la buena experiencia pasada con el TFG sabía que tendría que repetir si se me permitía. Muchas gracias por todo el interés mostrado y por esas respuestas a preguntas por Teams a todo tipo de horas.

También agradecer a mis padres y novia por todo el apoyo brindando a lo largo de todo el máster, sin vosotros no habría sido capaz.

Por último, también agradecer al grupo de trabajo encargado del proyecto IoT-Flock. Con el que se ha podido validar de una forma muy adecuada todo el desarrollo del proyecto.

David Sanches Gómez

Febrero 2021

Índice general

Índice de Figuras

Índice de Tablas

Glosario de acrónimos

- **ACK**: Acuse de recibo *(Acknowledgement).*
- **CoAP**: Protocolo de Aplicación Limitado *(Constrained Application Protocol).*
- **CON**: Confirmable *(Confirmable).*
- **HTTP**: Protocolo de Transferencia de Hipertexto *(Hypertext Transfer Protocol).*
- **IP**: Protocolo de Internet *(Internet protocol).*
- **IoT**: Internet de las Cosas *(Internet of Things).*
- **IIoT**: Internet de las Cosas Industrial *(Industrial Internet of Things).*
- **M2M**: Máquina a Máquina *(Machine to Machine).*
- **MQTT**: Protocolo de Transporte de Telemetría por Cola de Mensajes *(Message Queue Telemetry Transport).*
- **MQTT_SN**: Protocolo de Transporte de Telemetría por Cola de Mensajes para Redes de Sensores *(Message Queue Telemetry Transport for Sensor Networks).*
- **NON**: No-Confirmable *(Non-confirmable).*
- **OSI**: Modelo de Interconexión de Sistemas Abiertos *(Open System Interconnection).*
- **QoS**: Calidad de Servicio *(Quality of Service).*
- **RST**: Reinicio *(Reset).*
- **RAM**: Memoria de acceso aleatorio *(Random Access Memory).*
- **ROM**: Memoria de solo lectura *(Read-Only Memory).*
- **SQL**: Lenguaje de Consulta Estructurada *(Structured Query Language).*
- **TCP**: Protocolo de Control de Transmisión *(Transmission Control Protocol).*
- **UDP**: Protocolo de Datagramas de Usuario *(User Datagram Protocol).*
- **URI**: Identificador de Recursos Uniforme *(Uniform Resource Identifier).*

1 Introducción

1.1. Motivación

Con el paso de los años y el avance de la tecnología, el desarrollo de dispositivos conectados ha ido en aumento, introduciendo características y mecanismos de comunicación a todo tipo de objetos. Con ello, aparece lo que comúnmente se denomina como Internet de las Cosas (*Internet of Things*, IoT) [1].

En pocas palabras, IoT se refiere a la inclusión de elementos de comunicación a todo tipo de dispositivos físicos, que pueden ir desde un electrodoméstico a un sensor dentro de un termostato inteligente [2]. Estos sistemas, son capaces de recibir y transmitir datos a través de redes inalámbricas sin ningún tipo de intervención y, debido a la diversidad de dispositivos donde pueden estar implementados, se necesitan protocolos de comunicación adaptados a esta situación. Esto es necesario, ya que las limitaciones de *hardware* dependerán de cada dispositivo y se necesitarán protocolos que funcionen de igual manera en todo el sistema.

Por ello, aparecen protocolos diseñados específicamente para dispositivos IoT, cuyo objetivo principal es permitir la comunicación en los dispositivos con pocos recursos. Dos de los más importantes son el Protocolo de Aplicación Limitado (*Constrained Application Protocol*, CoAP) [3] y el protocolo de Transporte de Telemetría por Cola de Mensajes (*Message Queue Telemetry Transport*, MQTT) [4].

No obstante, debido a que estos protocolos son muy específicos, no cuentan con las numerosas herramientas de monitorización que podemos encontrar para protocolos más tradicionales (por ejemplo, HTTP), generando un gran vacío en el mercado, el cual se ve claramente necesitado de herramientas especificas.

Este Trabajo Fin de Máster propone la realización de un sistema que sea capaz de analizar los dos protocolos anteriormente mencionados, con el fin de poder generar registros de su tráfico de cara a monitorizar la red. Se pretende conocer el estado de la red y poder determinar el comportamiento de los sistemas que se encuentran funcionando

sobre ella y con ello, evaluar si su uso está siendo el esperado o incluso detectar posibles ataques maliciosos.

1.2. Objetivos

El objetivo de este Trabajo Fin de Máster, ha sido el desarrollo de un sistema de monitorización de tráfico IoT, más específicamente de los protocolos MQTT y CoAP.

- Se necesita comprender la estructura y funcionamiento de estos protocolos.
- Una vez estudiada, se debe realizar una herramienta que procese el tráfico.
- Con el tráfico procesado, se deberá seleccionar los campos que se consideren relevantes para la monitorización del tráfico, generando registros.
- Estos registros, deberán de ser plasmados de una manera gráfica. Con ello se logra facilitar el análisis de la red, ofreciendo información que ayude a conocer el estado de la misma.
- No se debe dejar a un lado, la necesidad de usar herramientas con las que simular un sistema IoT, de manera que se pueda validar la efectividad y el correcto funcionamiento del desarrollo realizado.

1.3. Fases de realización

La realización de este Trabajo Fin de Máster se ha dividido en las siguientes tareas, como se puede ver en el diagrama de Gantt de la **Figura 1.1**.

- **Estudio del estado del arte:** Durante esta tarea, se realizó la lectura de múltiples *papers*, artículos y estándares, con el fin de conocer en profundidad los protocolos IoT de CoAP y MQTT, así como, buscar y realizar un pre-análisis de las herramientas de generación de tráfico IoT. A su vez, se estudió el uso y funcionamiento de herramientas de procesamiento de datos como Grafana.
- **Análisis de casos de uso y requisitos:** Se plantearon los distintos casos de uso que se encuentran para este proyecto y con ellos se establecieron los distintos requisitos que debía tener el sistema de monitorización de redes IoT.
- **Diseño y Desarrollo:** En esta etapa, se implementó el programa que se encarga de monitorizar ambos protocolos, diseccionando los paquetes capturados durante un tráfico de datos. Con estos datos, se genera unos registros que posteriormente se han representado de manera gráfica.
- **Validación:** En esta fase, se ha llevado a cabo la comprobación del funcionamiento del código desarrollado, verificando su correcto funcionamiento y saneando los errores detectados. También, se han hecho pruebas con diferentes tipos de tráfico, de cara a validar el sistema como posible herramienta de detección de anomalías. Asimismo, se han verificado la disección del tráfico con herramientas como Wireshark.

- **Documentación:** Por último, se redactó la presente memoria, con el fin de plasmar el aprendizaje y trabajo realizado durante el desarrollo del proyecto. En ella, se explican todas las tareas realizadas y se exponen las soluciones llevadas a cabo, para obtener los resultados que en este documento se muestran.

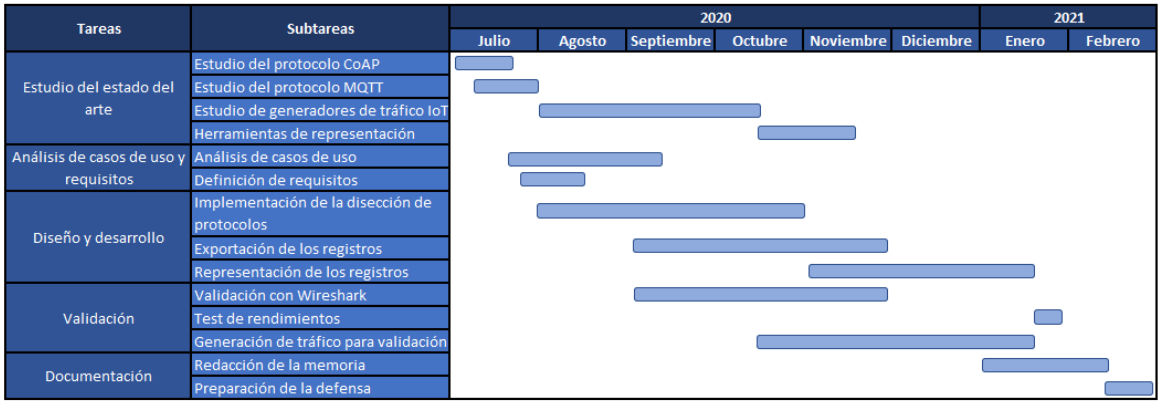

Figura 1.1: Diagrama de Gantt.

1.4. Estructura del documento

El resto de la memoria consta de los siguientes capítulos:

- **Estado del Arte:** A lo largo de este capítulo se explicará el concepto de IoT, centrándonos en los protocolos MQTT y CoAP, de los cuales se detalla su funcionamiento y estructura. Se describen los distintos proyectos donde se han encontrado ciertas funcionalidades que puedan asemejarse a las deseadas en este Trabajo Fin de Máster. Y, también, se comenta acerca de las distintas herramientas que permiten la generación de tráfico de estos protocolos y las bases de datos utilizadas, para procesar los datos y desplegarlos gráficamente en herramientas de visualización.
- **Análisis de Requisitos:** Una vez puesto en contexto al lector, se detallan las funciones que se espera que tenga el sistema a realizar, así como, los requisitos a cumplir.
- **Diseño y Desarrollo:** En este capítulo, se describen los pasos llevados a cabo para el desarrollo del sistema de monitorización. Comenzando por el desarrollo del analizador de protocolos, seguido de la carga de datos a las bases de datos y terminando con la representación gráfica de los registros.
- **Validación y pruebas:** Tras la finalización del desarrollo, se especifica todo el proceso llevado a cabo de cara a la validación del Trabajo Fin de Máster. A su vez, se explica el funcionamiento de uso de herramientas utilizadas para la validación del mismo.
- **Conclusiones y Trabajo Futuro:** Por último, se finaliza el documento detallando las conclusiones obtenidas con el desarrollo del trabajo, así como, las posibles mejoras que se pueden realizar de cara a la mejora del proyecto, por si se quisiera aumentar las funcionalidades de este.

2

Estado del Arte

2.1. Introducción

Durante este capítulo, se plasmará todos los conocimientos necesarios que se han requerido para el planteamiento y realización de este proyecto. Con ello, se pretende explicar de forma clara, toda la información necesaria para poder entender el desarrollo de este Trabajo Fin de Máster, de cara a facilitar la comprensión al lector.

En él, se explicará el concepto IoT, también se hablará de los dos protocolos tratados en este trabajo (MQTT y CoAP) [5, 6], desglosando su estructura y funcionamiento. A su vez, se hablará de diferentes herramientas para generar tráfico IoT y los programas de procesamiento de datos, de cara a una monitorización de los mismos.

2.2. IoT

IoT, cuyas siglas en inglés significan Internet de las Cosas (*Internet of Things*), implica la introducción de elementos de comunicación de red inalámbrica a todo tipo de dispositivos. Es decir, dotar a cualquier tipo de objeto de sensores, *software* y otro tipo de tecnologías, de manera que permita la conexión e intercambio de información con otros dispositivos, dentro de una red, a través de internet [7].

Gracias a la facilidad y sencillez de implantación de este tipo de tecnologías, hace que se encuentren presentes en todo tipo de objetos. Partiendo desde una simple bombilla dentro de una casa, hasta herramientas industriales complejas, como podemos ver en la **Figura 2.1** [8]. Poniendo un ejemplo, consideraríamos una red IoT a un conjunto de sensores, como pueden ser un sensor de humedad, otro de temperatura, etc, que ofrecen al usuario la información de los mismos, de manera autónoma de cara a tomar decisiones, por ejemplo, de activar el riego de un jardín.

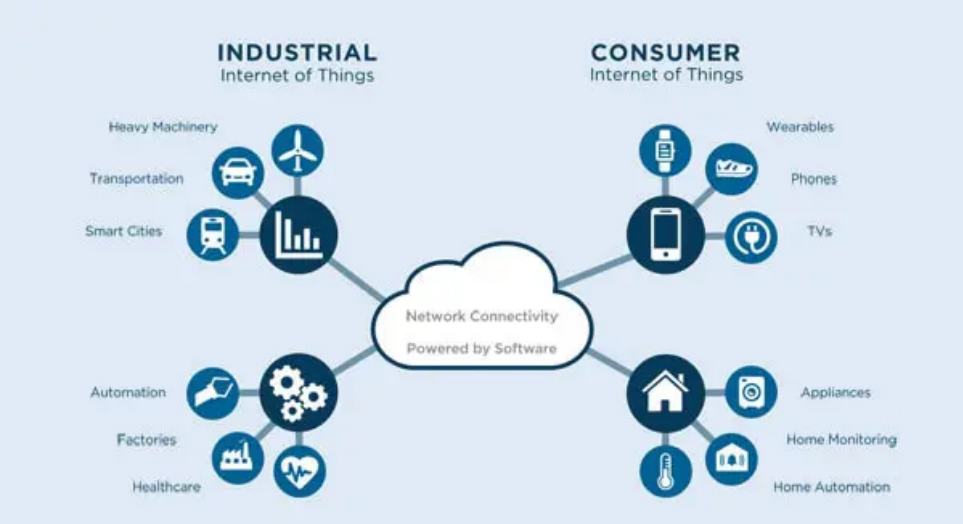

Figura 2.1: Diferencia entre IoT en la industria y para el consumidor [8].

Como se comenta en **Oracle** [2], la idea y el desarrollo del IoT existe desde hace mucho tiempo. No obstante, son ciertos avances tecnológicos los que han hecho posible que se permita poner en práctica de una manera útil y funcional.

Entre estos avances podemos destacar:

- La aparición de sensores de bajo coste y consumo eléctrico reducido permite que se introduzcan en toda clase de dispositivos y lugares [9].
- El desarrollo de protocolos específicos, que posibilitan la comunicación entre los dispositivos, como veremos en las **Secciones 2.3 y 2.4** [3, 4].
- El aumento de plataformas en las que se facilita que tanto las empresas como los consumidores puedan manejar la infraestructura IoT que posean, sin necesidad de invertir grandes cantidades de dinero y sin necesidad de pensar en un escalado futuro. Junto a ello, estas plataformas de *cloud computing* permiten, ligado con los avances en el aprendizaje automático, que se puedan recopilar y manejar grandes cantidades de datos por parte de los usuarios [10].
- También, destaca el gran auge que se está produciendo en la actualidad alrededor del aprendizaje automático, haciendo que dispositivos como Alexa, Siri, Google Home entre otros, cuenten con constantes mejoras, fomentando la demanda del gran publico de dispositivos conectados y, por ende, empujando a la industria a desarrollar nuevas tecnologías IoT de cara al consumidor [11].

No obstante, el gran valor y potencial del IoT se halla en la industria, con lo que se conoce como *Industrial IoT* o con sus siglas IIoT. Este, como se puede intuir, se refiere a la aplicación de tecnologías IoT en el entorno industrial, especialmente localizado en forma de instrumento de medida y control de sensores y equipos. Hasta ahora, en el terreno industrial, la comunicación entre los dispositivos se basaba en la comunicación máquina a máquina (*Machine to Machine*, M2M), pero gracias a la aparición de la nube y el aprendizaje automático, la industria obtiene nuevas formas de automatización nunca vistas hasta ahora, permitiendo nuevos modelos de negocio y mejoras en el control de las tareas realizadas [12].

2.3. MQTT

Como se ha planteado previamente, la diversidad de dispositivos en los que se puede encontrar IoT, con sus respectivas limitaciones, tanto de capacidad de cómputo como de alimentación, hace que se requiera de protocolos especializados en este ámbito. Con ello, surge el protocolo de Transporte de Telemetría por Cola de Mensajes (*Message Queue Telemetry Transport*, MQTT) [4].

MQTT es un protocolo de comunicación cuyo funcionamiento se basa en la transmisión de mensajes mediante paquetes, entre los publicadores y los subscriptores, haciendo uso de un servidor, este último denominado *broker*.

Al hacer uso del modelo publicación y subscripción, un cliente se suscribe a un intermediario o *broker* mediante un tema o *topic*. Este *topic*, sirve como medio de filtro para el *broker*, el cual reenviará la información que venga hacia él mediante mensajes de publicación (*Publish*) que posean el mismo *topic* como argumento. Un ejemplo básico de ello se puede apreciar en la **Figura 2.2**.

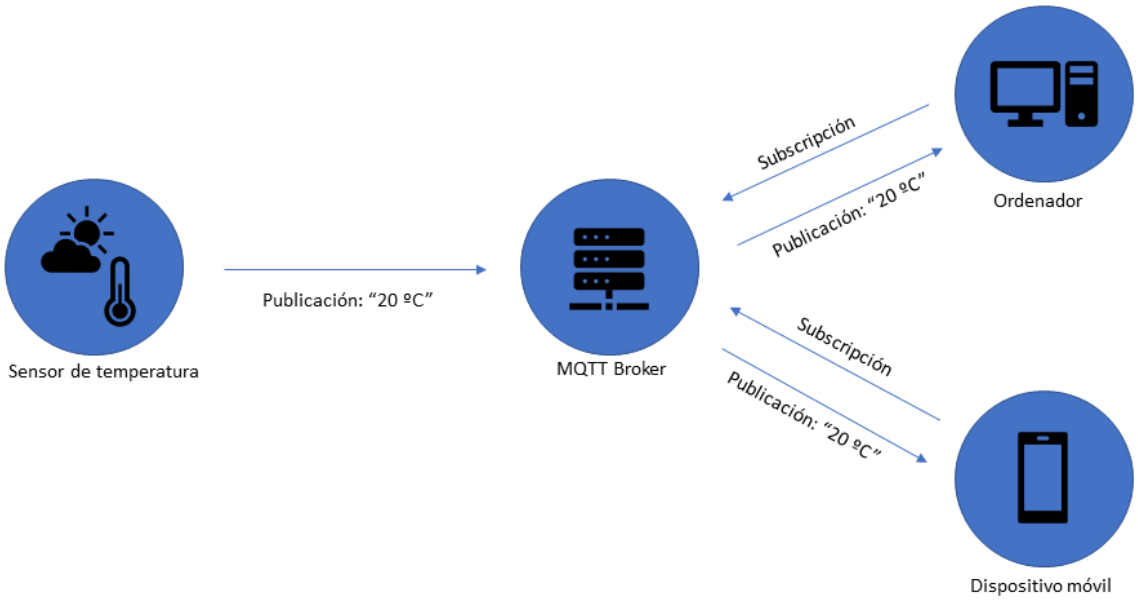

Figura 2.2: Ejemplo de funcionamiento de MQTT.

Como podemos ver, el funcionamiento de la anterior figura sería el siguiente:

- Primeramente, tanto el ordenador, como el dispositivo móvil, han mandando un mensaje de subscripción (*Subscribe*) hacia el *broker*, indicando el *topic* en el que previamente el sensor de temperatura se ha subscrito.
- Una vez hecho esto, el ordenador y el móvil recibirán la información del sensor de temperatura, el cual en este caso envía un mensaje indicando que la temperatura es de 20 °C. Será el *broker* el encargado de reenviar dicha información. Por lo tanto, nunca hay comunicación directa entre los diferentes clientes, todo viene del *broker*.

Como ya comprobaremos más adelante, el protocolo MQTT funciona sobre los protocolos IP y TCP de las capas L3 y L4 respectivamente, en el modelo de capas *OSI*. Siendo la L2 variable según la implementación [13], no obstante, en este trabajo se ha estudiado con la cabecera *Ethernet*, como vemos en la **Figura 2.3**.

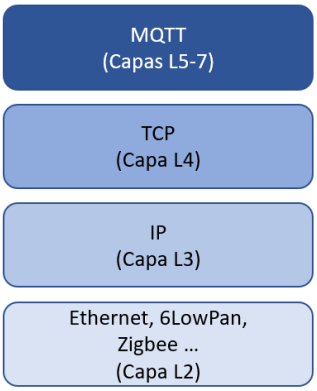

Figura 2.3: Formación de las capas de un paquete que cuenta con cabecera MQTT.

Cabe también mencionar una variante de este protocolo, con nombre MQTT_SN (Message Queue Telemetry Transport for Sensor Networks). Se caracteriza por estar desarrollado específicamente para funcionar en redes con sensores principalmente inalámbricos. La principal diferencia frente a MQTT es que funciona sobre UDP, e incorpora un nuevo elemento entre el sensor y el *broker*, conocido como MQTT_SN *Gateway* o pasarela. Esta última sirve para realizar la «traducción» entre el protocolo MQTT que sigue usando el *broker* y MQTT_SN que usa en estos casos el sensor, como se puede ver en la **Figura 2.4**.

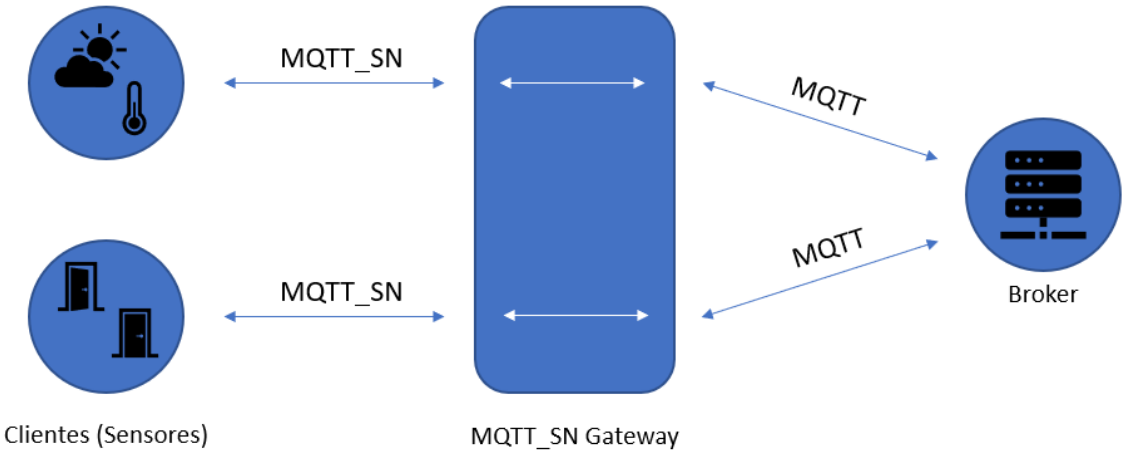

Figura 2.4: Ejemplo de funcionamiento de una red con protocolo MQTT_SN.

Como se puede apreciar en la **Figura 2.4**, una vez pasado el MQTT_SN *Gateway*, la red usa el protocolo MQTT tradicional, y dado que se entiende que la gestión de la red se hará del lado del broker, continuamos la explicación de este protocolo sin entrar en las cuestiones específicas de MQTT_SN.

En cuanto a la formación del mensaje MQTT, cuenta con una estructura formada por 3 partes, siempre en el mismo orden, como se aprecia en la **Figura 2.5**. Todo ello, se

encuentra explicado con mayor detalle en la pagina oficial de MQTT[1], donde encontraremos la información relativa a todas las versiones del protocolo. Cabe mencionar que, en este trabajo, nos hemos centrado en la versión 3.1.1 [4], debido a que es la que tiene mayor implantación [14] y es la que podemos generar con la herramienta IoT-Flock, que veremos en la **Sección 5.3.2**.

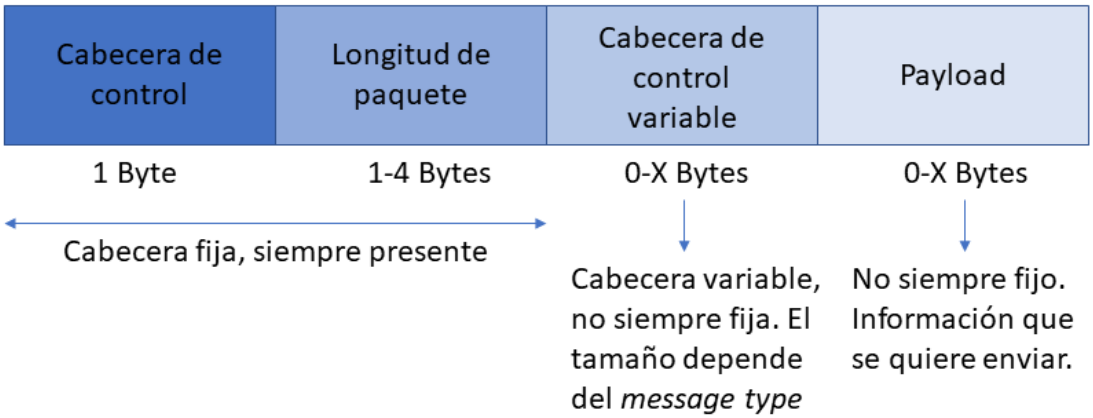

Figura 2.5: Estructura general de un paquete MQTT [15].

Fijándonos en la figura anterior, podemos observar la cabecera fija, la cual siempre está presente y se compone de entre 2 a 5 bytes, dependiendo del valor de la variable de longitud de paquete que explicaremos más adelante. Está formada por la cabecera de control, compuesta por 1 byte y la variable de longitud de paquete, formada por el número de bytes que ocupa la escritura del número en binario. Hay que destacar, que esta representación se hace haciendo uso únicamente de 7 bits, ya que el 8° bit es el que indica si se va a hacer uso de otro byte extra para la representación del número. Por lo tanto, el máximo valor que se puede representar con un solo byte es 127, y si es superior se necesita hacer uso del siguiente byte, estableciendo el bit más significativo a 1, como vemos en la **Tabla 2.1**. Esta variable longitud de paquete, indica el número de bytes que ocupa la cabecera de control variable más el *Payload*.

Tabla 2.1: Número de bytes que ocupa la variable longitud de paquete.

Número de bytes	**Desde**	**Hasta**
1	0 (0x00)	127 (0x7F)
2	128 (0x80, 0x01)	16383 (0xFF, 0x7F)
3	16384 (0x80, 0x80, 0x01)	2097151 (0xFF, 0xFF, 0x7F)
4	2097152 (0x80, 0x80, 0x80, 0x01)	268435455 (0xFF, 0xFF, 0xFF, 0x7F)

Adentrándonos en la cabecera de control, esta está formada por las variables *Packet type*, que ocupa los 4 primeros bits, y la variable *Flags*, rellenando los otros 4. En cuanto a los tipos de *Packet type*, estos se encuentran en la **Tabla 2.2**, indicando su nombre, el valor en decimal en el que se encontrarían en el paquete, la dirección donde puede operar y una pequeña descripción de su significado.

[1] `https://mqtt.org/mqtt-specification/`

Tabla 2.2: *Packet type* de un paquete MQTT.

Nombre	Número	Dirección	Descripción
CONNECT	1	Cliente al Servidor	Petición del cliente de conectarse al servidor
CONNACK	2	Servidor al Cliente	ACK Asentimiento de la conexión
PUBLISH	3	Cliente al Servidor Servidor al Cliente	Publicación del mensaje
PUBACK	4	Cliente al Servidor Servidor al Cliente	ACK de la publicación
PUBREC	5	Cliente al Servidor Servidor al Cliente	Publicación recibida (entrega asegurada, parte 1)
PUBREL	6	Cliente al Servidor Servidor al Cliente	Publicación enviada (entrega asegurada, parte 2)
PUBCOMP	7	Cliente al Servidor Servidor al Cliente	Publicación completada (entrega asegurada, parte 3)
SUBSCRIBE	8	Cliente al Servidor	Petición de subscripción por parte del cliente
SUBACK	9	Servidor al Cliente	ACK del SUBSCRIBE
UNSUBSCRIBE	10	Cliente al Servidor	Petición de desubscrpición por parte del cliente
UNSUBACK	11	Servidor al Cliente	ACK del UNSUBSCRIBE
PINGREQ	12	Cliente al Servidor	Solicitud de PING
PINGRESP	13	Servidor al Cliente	Respuesta de PING
DISCONNECT	14	Cliente al Servidor	El cliente se desconecta

En cuanto a los *Flags* de la cabecera fija, el valor de estos depende del tipo de *Packet type* que contenga el paquete. De manera general, los 4 bits (posiciones de bit 3, bit 2, bit 1 y bit 0) suelen ser 0, a excepción de los paquetes de tipo *PUBREC*, *SUBSCRIBE* y *UNSUBSCRIBE*, los cuales cuentan con el bit de la posición 1 con un valor de 1. Y en el caso de los paquetes de tipo *PUBLISH*, en la posición de bit 3 cuentan con la variable *Duplicate delivery flag* (DUT) la cual indica, si se encuentra a 1, que el mensaje es una «re-publicación». En cuanto a las posiciones de bit 2 y 1, se indican la QoS (*Quality of Service*) pudiendo ser 0, 1 o 2. El significado de cada uno lo podemos ver en la **Tabla 2.3**. Finalmente, el último bit, que se encuentra en la posición 0, es el bit de la variable *Retain*. Esta variable, si se encuentra fija a 1, indica que el mensaje *PUBLISH* enviado por el cliente debe de ser guardado por el *broker*. Un ejemplo de esto último, puede darse en el siguiente escenario: si un sensor indica su estado únicamente cuando cambia, por ejemplo una puerta abierta o cerrada, si un nuevo subscriptor se subscribe al *topic*, no habría ninguna manera de conocer su estado, de no ser porque la variable *Retain* esté activa y se guarde su estado en el *broker* [16].

Tabla 2.3: Variable QoS de un paquete MQTT.

Valor QoS	Bit 2	Bit 1	Descripción
0	0	0	Como mucho llega un PUBLISH
1	0	1	Como mínimo llega un PUBLISH
2	1	0	LLega exáctamente un PUBLISH

En cuanto al resto de parámetros, estos son dependientes del tipo de *Packet type* que se envíe. En este documento, mencionaremos aquellos que se han considerado relevantes de cara al desarrollo del proyecto, pero si se quiere más información del resto de variables, se recomienda visitar la pagina oficial de MQTT referenciada con anterioridad.

Comenzamos con los parámetros de los paquetes de tipo CONNECT que, como hemos visto en la **Tabla 2.2**, son los que indican la petición de inicio de sesión desde el cliente

hacia el *broker*. Este tipo de paquete sólo puede enviarse una vez por cliente y si se envía más de uno, el *broker* suspenderá la conexión con el cliente.

En cuanto a las variables que hemos considerado importantes, cabe destacar el *Protocol Version*, el cual se encuentra en la parte variable de la cabecera MQTT. Esta variable representa la versión MQTT que se esté utilizando en la transmisión. Como referencia, en las pruebas realizadas en este proyecto hemos visto dos tipos de versiones: la versión 3.1, que toma como valor 3 y la versión 3.1.1 con valor 4.

Otro parámetro que también se ha utilizado en este desarrollo, perteneciente a los paquetes de tipo CONNECT, es el *Client Identifier*. Como su propio nombre indica, sirve para distinguir a los clientes por parte del servidor, ya que cada cliente que se conecte debe de tener asignado un *string* de *ClientId* único. Este campo es de obligada aparición y se debe localizar como primer campo de la zona de *Payload* del paquete CONNECT.

También mencionar el parámetro *User Name*, el cual aparecerá siempre y cuando la variable de *User Name Flag* se encuentre a 1. Cuando aparece esta variable, puede servirle al *broker* como método de autenticación y autorización.

Una vez que el inicio de conexión se ha realizado por parte del cliente, el *broker* debe de responder con un CONNACK, en el cual la variable *Return Code* indica si la conexión es aceptada o si ha ocurrido un rechazo por algún motivo, como podemos apreciar en la **Tabla 2.4**.

Tabla 2.4: Tipos de *Return Code* [17].

Return Code	**Respuesta**
0	Conexión aceptada.
1	Conexión rechazada, versión del protocolo no válida.
2	Conexión rechazada, identificador rechazado.
3	Conexión rechazada, servidor no disponible.
4	Conexión rechazada, usuario o contraseña erróneos.
5	Conexión rechazada, no autorizado.

Podemos comentar también el parámetro *Topic*, en el que se mostrará el tipo de petición que se está realizando en un mensaje de tipo PUBLISH.

2.4. CoAP

Otro protocolo que hemos tratado en este trabajo es el Protocolo de Aplicación Limitado (*Constrained Application Protocol*, CoAP) [3], el cual surge, al igual que MQTT, por la necesidad de tener medios de comunicación, *Machine to Machine*, en dispositivos con pocos recursos de ROM y RAM, en medios con muchas perdidas de paquetes o redes con alta congestión, funcionando por lo tanto en momentos donde modelos basados en TCP, como MQTT, no podrían funcionar de manera adecuada.

Su funcionamiento es similar a HTTP en el que se basa, operando sobre UDP y poseyendo los mismos métodos que HTTP, pero trabajando de forma asíncrona. Se caracteriza por hacer uso del modelo petición-respuesta. Por lo tanto, el cliente es el que realiza la

petición, usando un *Method Code*, a un recurso que se encuentre en el servidor. A este recurso se accede por medio de una *URI*. A continuación, el servidor envía la respuesta con un *Response Code*.

En cuanto a los tipos de mensajes que usa CoAP, podemos ver 4 tipos:

- *Confirmable*, CON.
- *Non-confirmable*, NON.
- *Aknowledgment*, ACK.
- *Reset*, RST.

Y en ellos se hace uso de 4 tipos de métodos, similares a los de HTTP:

- *GET*.
- *PUT*.
- *POST*.
- *DELETE*.

Podemos establecer que la cabecera de CoAP se define en dos partes, la más baja denominada *CoAP Messages Model* y la segunda capa con nombre *CoAP Request/Response Model*.

La primera capa es la responsable de hacerse cargo del intercambio de mensajes sobre UDP entre las entidades. Cada mensaje de CoAP usa una cabecera fija, formada por 4 bytes, que puede estar acompañada por unas opciones en forma binaria y su respectivo *payload*. También, cabe mencionar que cada mensaje contiene un identificador único, de manera que se puedan detectar mensajes duplicados.

Como se ha dicho previamente, hay dos tipos de mensajes: están los que requieren confirmación por parte del receptor, *Confirmable*, y los que no necesitan, *Non-confirmable*. Estos primeros, al necesitar que el servidor responda para confirmar su recepción, están continuamente enviando el mensaje hasta que el receptor envíe la confirmación de llegada, mediante un mensaje de tipo *Aknowledgment*. Este mensaje de vuelta, debe de contener el mismo identificador que el enviado. De esta manera, se comprueba su correcta recepción. Por otro lado, los que no necesitan confirmación del receptor, por ejemplo, el envió periódico de información de un sensor, hacen uso de mensajes *Non-confirmable*. Al igual que con los *Confirmable*, se envían con un identificador de mensaje, para detectar los duplicados. Y si el receptor no es capaz de procesarlos se suele responder con un mensaje de tipo *Reset*.

En cuanto a la segunda capa, la formada por el modelo de petición/respuesta (*CoAP Request/Response Model*), se pueden distinguir diferentes situaciones dependiendo de si el servidor puede o no responder de manera inmediata a la petición solicitada, que puede llegar de forma de un mensaje *Confirmable* o *Non-confirmable*.

Si el servidor puede responder de manera inmediata, la respuesta la puede enviar directamente dentro del mensaje de tipo ACK, si el mensaje enviado es de tipo *Confirmable*,

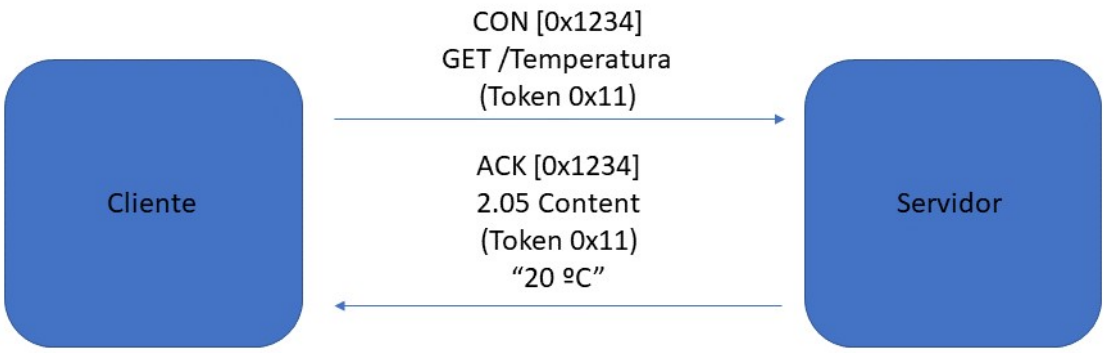

Figura 2.6: Ejemplo de intercambio de mensajes en CoAP con respuesta inmediata.

como podemos ver en la **Figura 2.6**, o con un *Non-confirmable* si el mensaje no requiere de confirmación.

En la figura anterior, aparece la variable *Token*, hasta ahora no mencionada y diferente al identificador de mensaje ya comentado previamente. Esta variable se usa para hacer coincidir una respuesta con una solicitud. Todos los mensajes deben llevarlo y las respuestas deben de usar el mismo *Token* que la solicitud a la que atienden. La finalidad de este parámetro es la de servir como identificador cliente-local para diferenciar entre solicitudes concurrentes.

Por el contrario, cuando el servidor no es capaz de enviar la respuesta de forma inmediata, lo que hará es, en el caso de mensajes de tipo *Confirmable*, enviar primero una respuesta ACK como un mensaje vacío y posteriormente enviar un mensaje *Confirmable* con el contenido de la respuesta. En estos casos, el cliente responderá con un mensaje ACK para confirmar la recepción de la información, como se representa en la **Figura 2.7**

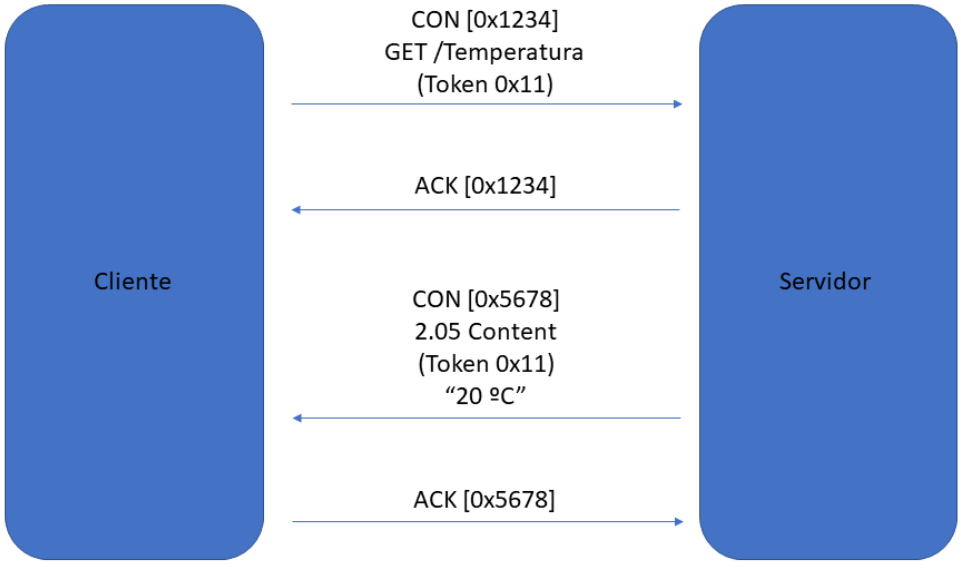

Figura 2.7: Ejemplo de intercambio de mensajes en CoAP sin respuesta inmediata.

Una vez analizado el funcionamiento del protocolo, nos centramos en cómo está compuesto un mensaje CoAP. Al igual que ocurre con MQTT, CoAP posee ciertos parámetros que son fijos en todo paquete y otras variables que aparecen dependiendo del tipo de mensaje que se envíe.

La cabecera fija comienza con los 2 primeros bits, indicando la versión CoAP que se está usando, seguido de los dos siguientes bits, los cuales indican si el tipo de mensaje que se está enviando es de tipo CON, NON, ACK o Reset con los números 0, 1, 2 y 3 respectivamente en binario. Los 4 siguientes bits que conforman el primer byte de la cabecera, vienen determinados por la variable *Token Length* (TKL), indicando la longitud del campo *Token* (0-8 bytes). El siguiente byte indica el campo *Code*, el cual se corresponde con el código de respuesta o petición que se está enviando en el mensaje. En la **Tabla 2.5** podemos ver las distintas alternativas de códigos que tenemos en un paquete.

Tabla 2.5: Métodos de un paquete CoAP.

Code	**Descripción**
0.01	GET
0.02	POST
0.03	PUT
0.04	DELETE
2.01	Creado
2.02	Eliminado
2.03	Validado
2.04	Cambiado
2.05	Contenido
4.00	Mala petición
4.01	No autorizado
4.02	Opción errónea
4.03	Prohibido
4.04	No encontrado
4.05	Método no permitido
4.05	No aceptable
4.06	Predicción fallida
4.12	Acondicionamiento fallido
4.13	Entidad solicitada demasiado grande
4.15	Formato de contenido no soportado
5.00	Error interno del servidor
5.01	No implementado
5.02	Puerta de enlace incorrecta
5.03	Servicio no disponible
5.04	Tiempo de espera de puerta de enlace acabado
5.05	Proxy no soportado

Se prosigue con el *Message ID* el cual, formado por 2 bytes, identifica cada mensaje para así poder detectar duplicados, como hemos mencionado con anterioridad. A continuación, aparece la variable *Token*, la cual estará formada desde 0 a 8 bytes, dependiendo

del valor indicado en la variable *Token Length field*. Como se ha dicho en párrafos anteriores, sirve para correlacionar las peticiones y las respuestas. Por último, aparecen los campos de opciones, los cuales pueden estar o no, al igual que el *payload*.

2.5. Monitorización IoT

Una vez explicado el termino IoT (**Sección 2.2**), así como, los protocolos más usados en este ámbito (**Secciones 2.3 y 2.4**), nos centramos en la idea de la monitorización de estos sistemas.

Dado que una red IoT suele estar formada por largas cantidades de dispositivos, los cuales suelen enviar datos de forma periódica y estar localizados de manera muy dispersa, se puede intuir la dificultad que surge de cara a conocer el estado de funcionamiento de esta red, así como, el estado de los dispositivos de ella. Por ello, surge la necesidad de contar con herramientas que permitan controlar el estado de salud de la red, detectar posibles funcionamientos erróneos de los dispositivos o la falta de tráfico de estos, e incluso descubrir posible ataques.

Para todo ello, existen en el mercado ciertas herramientas que pueden ayudar a esta tarea. Una de ellas es ndpi de ntop [18], herramienta que permite la detección de tráfico para su clasificación según el tipo de protocolo. No obstante, esta herramienta únicamente sirve para detectar, en el caso de IoT, que se está haciendo uso de los protocolos MQTT y CoAP, por lo tanto no da ningún tipo de información acerca del estado de la red, ni de la información transportada por los paquetes.

Por otro lado, podemos ver una implementación realizada por un grupo de investigación de la Universidad Tecnológica de Brno [19]. En ella, su objetivo era el de realizar modelos de predicción de tráfico MQTT y CoAP de manera que fueran capaces de detectar incidentes de seguridad. Para ello, necesitaban primero extraer los datos de los paquetes transmitidos. Una vez extraído, proponen dos técnicas para analizar el flujo de registros: una se basa en consultar la base de datos de registros de flujo IoT y compararla con la extraída en el flujo capturado. La segunda propuesta, se basa en la detección de anomalías de manera estadística comparando los resultados a analizar con los ya obtenidos a largo plazo en ese sistema. Esta propuesta, sin embargo, se centra únicamente en mostrar, de una manera muy manual, si se están produciendo algún tipo de anomalía del estilo a un ataque. No muestra, de una manera gráfica y *user friendly* el estado de salud de la red, ni lo que puede estar ocurriendo, ni tampoco el donde. Además, cabe mencionar que no se encuentra disponible el código completo de forma abierta.

Adicionalmente a los trabajos anteriores, también podemos hablar del sistema de monitorización de flujos IoT realizado por un grupo de investigación portugués [20]. La idea de ese estudio es muy similar a la planteada en este Trabajo Fin Máster. Se comienza con el análisis del tráfico capturado, MQTT y CoAP, del cual se extraen los flujos, que posteriormente se exportan a un colector de flujos, al cual se accede para realizar la monitorización. No obstante, el uso de flujos para la monitorización de la red conlleva ciertas consecuencias, siendo la más importante la pérdida de resolución. Dado que se agrupa el tráfico siguiendo los flujos, se pierde la información de la capa de aplicación ligada a cada tipo de paquete transportado. Sin embargo, se reduce de forma considerable el tamaño que ocupa la base de datos.

Se encuentra, por tanto, un claro vacío en este ámbito, que trata de cubrir este Trabajo Fin de Máster al plantear el desarrollo un sistema que monitorice de forma precisa el tráfico IoT. Por lo tanto, se ha implementado una herramienta que procesa el tráfico MQTT y CoAP y, posteriormente, lo representa de forma gráfica, de modo que se pueda dar a conocer el estado de la red o incluso detectar anomalías.

2.6. Generación de tráfico IoT

Una que vez se ha comentado cómo son los protocolos con los que se va a tratar en este Trabajo Fin de Máster, nos centramos en hablar de la generación de tráfico con estos protocolos. Este apartado es de gran importancia, debido a que ha sido de gran ayuda, de cara a la validación del proyecto, el contar con tráfico real o simulado de manera que se ha podido comprobar que los sistemas desarrollados funcionan con propiedad.

Dada la peculiaridad de los protocolos MQTT y CoAP, se necesitan implementaciones específicas para la generación de este tráfico.

Por ello, en esta sección nos centramos en analizar las diferentes implementaciones estudiadas y testeadas, para la generación de tráfico tanto real como simulado, destacando las siguientes que vienen a continuación.

2.6.1. Cooja

Cooja es un simulador de redes IoT cuyo funcionamiento recae en la programación que realiza este sobre Contiki [21]. Contiki es un sistema operativo, de código abierto, pensado para ser utilizado en la comunicación de dispositivos de baja potencia inalámbricos, es decir sistemas IoT.

Gracias al simulador Cooja, una vez esta instalado todo, se puede generar una red simulada con tráfico CoAP de manera que con ello se podrían tomar capturas que servirían para el desarrollo del proyecto. Se debe mencionar, que este tráfico funciona sobre la cabecera 6LoWPAN[2].

2.6.2. CoAP Shell

Prosiguiendo con herramientas de generación de tráfico CoAP, nos encontramos con CoAP Shell [22]. Este proyecto ofrece una interfaz interactiva, funcional mediante comandos, que permite interactuar con servidores CoAP. Funciona sobre un archivo ejecutable Java, por lo que su puesta en marcha es muy sencilla.

Para poder hacer uso de esta herramienta es necesario contar con un servidor con el cual se puedan hacer las peticiones de tráfico CoAP. Para esto, se testearon dos servidores de gran uso en este ámbito que son coap.me[3] y Eclipse Californium[4]. En ambos, se puede testear el funcionamiento de CoAP y generar tráfico, por lo que es una herramienta de gran interés para este proyecto, al igual que los servidores mencionados.

[2] `https://tools.ietf.org/html/rfc8138`
[3] `http://coap.me/`
[4] `https://www.eclipse.org/californium/`

2.6.3. Mosquitto

En este caso, nos encontramos con Mosquitto, un proyecto de código abierto que implementa un *broker* del protocolo MQTT, funcional en todas las versiones del protocolo.

Este *broker*, al estar pensado directamente para funcionar sobre redes IoT, cuenta con una implementación muy "ligera", pudiendo funcionar sobre dispositivos de poca potencia.

El uso que se ha dado a este proyecto recae sobre IoT-Flock (**Sub-sección 2.6.4**), herramienta que utiliza este *broker* para la generación de tráfico MQTT.

2.6.4. IoT-Flock

Por último, hablaremos de la herramienta que se ha considerado como la más útil para el desarrollo del proyecto. Nos referimos a IoT-Flock [23], desarrollado por un grupo de trabajo del *Al-Khawarizmi Institute of Computer* en Pakistán.

Se trata de un simulador de tráfico IoT que, más concretamente, es capaz de simular los dos protocolos tratados en este trabajo (MQTT y CoAP). Este *framework*, permite al usuario generar una serie de casos de uso, en los que se establecerán distintos dispositivos/sensores, que se encargarán de simular una red IoT, enviando información de manera periódica.

Esta herramienta, aparte de poder generar tráfico mediante la simulación de dispositivos, también permite la generación de tráfico malicioso, como pueden ser ataque por inundación. Esta característica es de gran interés, ya que no existe en este ámbito ninguna otra herramienta que permita simular este tipo de situaciones para los protocolos IoT que tratamos.

En el **Apartado 5.3.2** ampliaremos con más detalle la forma de uso de esta herramienta y todas las funciones que se han utilizado en el proyecto, así como los problemas que han surgido a la hora de hacer uso de esta.

2.6.5. NetSim

Hasta ahora se han tratado siempre herramientas de software libre que permiten la simulación de red, pero también contamos con herramientas de pago que pueden hacer esta tarea. Una de estas es NetSim, de la empresa Tetcos [24].

Con esta herramienta se puede diseñar, simular y analizar el rendimiento de redes IoT, gracias a que es capaz de reproducir el funcionamiento de dispositivos, enlaces, hacer uso de distintos tipos de encapsulado, etc. Sin embargo, dado a que este programa es de pago no se ha podido investigar profundamente su funcionamiento, ni validar su uso para este trabajo.

2.7. Herramientas de procesamiento de datos y representación

Como se ha ido mencionando, el propósito de este trabajo es primero extraer los parámetros, que consideramos interesantes, de cada uno de los protocolos y de ahí generar unos registros. No obstante, estos registros no son interpretables de forma eficiente sin una herramienta que sea capaz de obtener estos datos y posteriormente procesarlos, de cara a realizar una representación de los mismos que permita al usuario conocer la situación actual de la red en cuestión.

Para realizar esto, se ha hecho uso de la herramienta Grafana[5]. Esta, es una herramienta de código abierto, multiplataforma, usada para analizar archivos de *logs*, monitorización de datos en tiempo real, monitorización de aplicaciones, seguridad y mucho más, haciendo uso de la representación de los datos mediante unos paneles formado por gráficos, como podemos apreciar en la **Figura 2.8**.

Figura 2.8: Ejemplo de un panel de Grafana.

Cabe mencionar la existencia de otras herramientas similares a Grafana, como es Kibana[6]. Esta hace uso de Elasticsearch como base de datos y permite, al igual que Grafana, la representación gráfica de los datos, así como, manejar las distintas bases de

[5] https://grafana.com/
[6] https://www.elastic.co/es/kibana

datos que se encuentran en el servidor de Elasticsearch, permitiendo obtener información sobre ellas o incluso eliminarlas.

El modo de funcionamiento de Grafana es muy sencillo, cuenta con una interfaz gráfica muy intuitiva que va guiando al usuario de cara a la creación de los paneles. Para acceder a esta interfaz gráfica, se accede mediante el acceso de la IP local del dispositivo (si no se indica otra) y añadiendo el puerto 3000, que está definido por defecto.

Debido a la variedad de tipo de bases de datos disponibles para usar en Grafana (Elasticsearch, InfluxDB, Prometheus, MySQL, etc), durante el desarrollo aparece la duda de cuál usar, cual será la más adecuada. Por este motivo, se decide hacer uso de dos tipos de bases de datos, una de tipo SQL como puede ser MySQL, y otra no relacional, como sería Elasticsearch. La decisión de testear estas particularmente es su mayor uso por parte de los desarrolladores y, dado que las diferencias no son muy grandes entre ellas, no se ha visto en la necesidad de hacer pruebas con otras a mayores.

En el **Capítulo 4**, se explicará con más detalle el uso de cada una de estas bases de datos y como se ha hecho uso de ellas. Se ampliará la explicación del funcionamiento de Grafana y los pasos seguidos para la creación de los paneles.

2.8. Conclusiones

A lo largo de este capítulo, se han sentado las bases en las que se basa este Trabajo Fin de Máster, introduciendo el término IoT con el que damos a conocer el problema y los motivos por los cuales se ha optado a realizar este proyecto.

También se ha explicado el funcionamiento y la estructura básica de los paquetes de ambos protocolos tratados a lo largo del trabajo, MQTT y CoAP. Y se han introducido herramientas de monitorización IoT.

A su vez, hemos tratado el tema de los generadores de tráfico IoT, mostrando la gran importancia que tienen para la validación y se han introducido los principales que se han probado, dejando claro que IoT-Flock será el que usaremos en mayor medida.

Por último, se ha hecho una breve introducción de las bases de datos (MySQL y Elasticsearch) y a las herramientas de procesado datos que vamos a usar, como Grafana.

Todo esto unido permitirá que se pueda desarrollar el sistema de monitorización que describimos en los capítulos siguientes.

3

Análisis de Requisitos

3.1. Introducción

Una vez tratado los temas de IoT, los protocolos que se usan en él y los resultados que queremos obtener con el proyecto, nos centramos en dejar plasmado los distintos casos de uso que nos encontramos, de cara a poder definir los requisitos que debe cumplir este desarrollo para tomarlo como válido.

Por ello, se han dividido estos requisitos en dos aspectos, los requisitos funcionales, que son los distintos desempeños, resultados y características con los que debe contar el sistema y, los no funcionales, que son los criterios que definen las propiedades del sistema (comportamiento, eficiencia ...)[25], de cara al desarrollo y validación de las funcionalidades de este Trabajo Fin de Máster.

3.2. Casos de uso

Comenzamos desarrollando los casos de uso que podemos ver de cara a la realización del sistema de monitorización desarrollado. En la **Figura 3.1** se muestra el diagrama de casos de uso. Podemos apreciar un único actor, el gestor de red, encargado de evaluar, a partir de los resultados, la red y sensores que se encuentran funcionando.

En cuanto a los casos de uso planteados se encuentran:

- **Monitorización IoT:** proceso de monitorización del tráfico. Se comienza con la captura del tráfico MQTT o CoAP, para después procesar los datos y almacenarlos en bases de datos a las que se acudirá para la representación de los datos.
- **Rendimiento de la red:** se puede medir el rendimiento de la red, para obtener una valoración de la misma

- **Identificar anomalías:** se puede identificar anomalías en la red, como por ejemplo ataques.
- **Funcionamiento de sensores:** se puede comprobar el comportamiento de los sensores, verificando así posibles errores en los mismos.
- **Controlar conexiones:** se trata de identificar intentos de conexión no deseados, contrastando que todas las conexiones de la red siguen activas y no han dejado de funcionar.

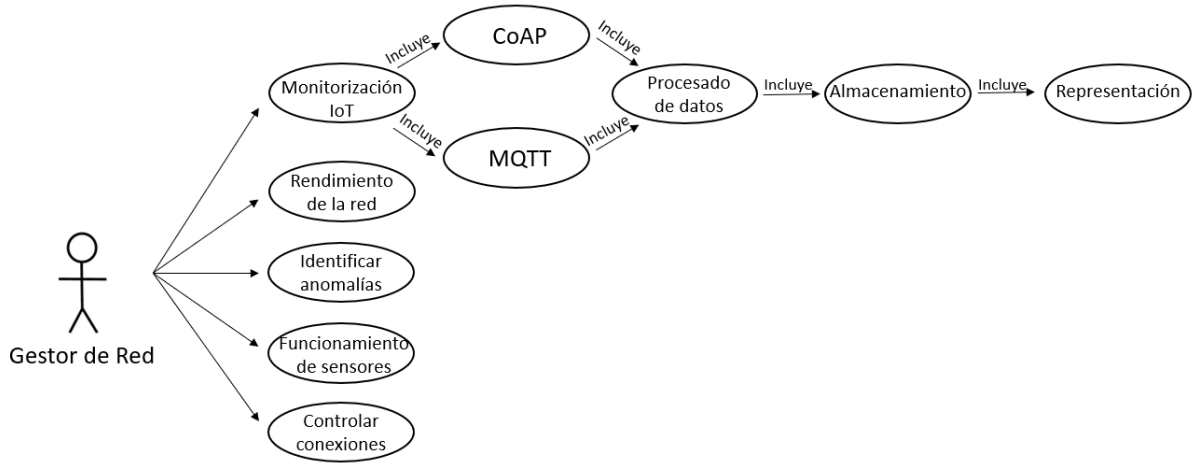

Figura 3.1: Casos de uso.

3.3. Requisitos funcionales

El sistema de monitorización de protocolos IoT cuenta con una serie de requisitos que se espera que se lleven a cabo en su funcionamiento, de cara a certificar que los objetivos impuestos a la hora de realizar el proyecto se cumplen.

El principal requisito de este proyecto es el de ofrecer una herramienta que ayude a la toma decisiones a la hora de manejar una red. Por lo tanto, el primer requisito que se debe cumplir, es la correcta obtención de los campos que conforman los paquetes capturados en la red IoT, los cuales deben usar uno de los dos protocolos ya mencionados (MQTT y CoAP). Un mal funcionamiento de esta etapa conformaría un error que se arrastraría a lo largo de todo el sistema, ya que conforma el «cimiento» donde se sustenta el resto del desarrollo.

Siguiendo con los requisitos funcionales, se requiere que los datos obtenidos de los paquetes, sean gestionados por una base de datos sin perder ninguno de estos parámetros. Por ello, se necesita que la configuración de inserción de estos se adecuada. Esto es debido a que una pequeña falla de la base de datos puede ocasionar mal interpretaciones de estos, dando lugar a decisiones incorrectas.

Otro requisito funcional es la visualización de los parámetros de una manera clara e intuitiva. Se necesita que los datos almacenados en las bases de datos puedan ser

representados de manera gráfica, de cara a lograr una rápida toma de decisión ante los datos mostrados por parte de un gestor de red.

También se incluye la idea de que este desarrollo no se utilice únicamente como método de monitorización del estado de la red, sino añadir un grado más de completitud, como herramienta de aviso de anomalías. Mediante el uso de alarmas, el sistema puede ser utilizado para poder detectar ataques que se estén produciendo en la red.

3.4. Requisitos no funcionales

Una vez vistos los requisitos funcionales, nos centramos en los no funcionales. Entre estos, destaca que el sistema sea realmente capaz de ofrecer un grado de soporte a los gestores de red que se encargan de manejar los sistemas de redes IoT.

Otro requisito va ligado al rendimiento. Debido a la naturalidad de estos sistemas, se requiere que el desarrollo tenga tiempos de ejecución bajos, ayudando así a un preprocesado rápido de los paquetes, de manera que se puedan añadir a la base de datos con la mayor rapidez posible, siendo el tiempo de procesado necesariamente inferior a la duración del tráfico capturado

A su vez, se requiere que el sistema sea intuitivo y de fácil manejo. Un gestor de red debe poder entender las representaciones de manera que sea capaz de tomar decisiones de manera instantánea.

De igual manera, se necesita que el sistema de monitorización tenga capacidad de evolucionar. Es decir, que se permita añadir nuevas funcionalidades o modificar las existentes, de cara a ofrecer mayor capacidad de control al gestor de red.

Por último, se necesita poder comprobar el correcto funcionamiento del sistema de monitorización. Se requiere que el sistema en su conjunto funcione sin fallos.

3.5. Conclusión

Gracias a este capítulo, se han especificado los casos de uso y requisitos que se plantean para la completitud de este Trabajo Fin de Máster. Se han dividido en requisitos funcionales, como puede ser el correcto funcionamiento del sistema de análisis de tráfico y, en no funcionales, como podría ser el grado de utilidad que otorga este proyecto a los analistas de redes IoT.

En el siguiente capítulo se comentarán todos los procesos realizados para el desarrollo del sistema de monitorización de redes IoT descrito.

4

Diseño y Desarrollo

4.1. Introducción

A lo largo de este capítulo se explicará todo el proceso llevado a cabo para la realización de este proyecto. Se detallarán las distintas etapas realizadas, siguiendo el mismo orden de funcionamiento del sistema.

Se comienza, como se puede ver en la **Figura 4.1**, con el análisis de los protocolos, extrayendo la información de los protocolos de manera que se pueda generar un archivo de registros. Con este archivo, se procederá a procesarlos y cargarlos en la base de datos y, a través de ella, se irá accediendo a la misma, mediante la visualización gráfica de los datos con Grafana.

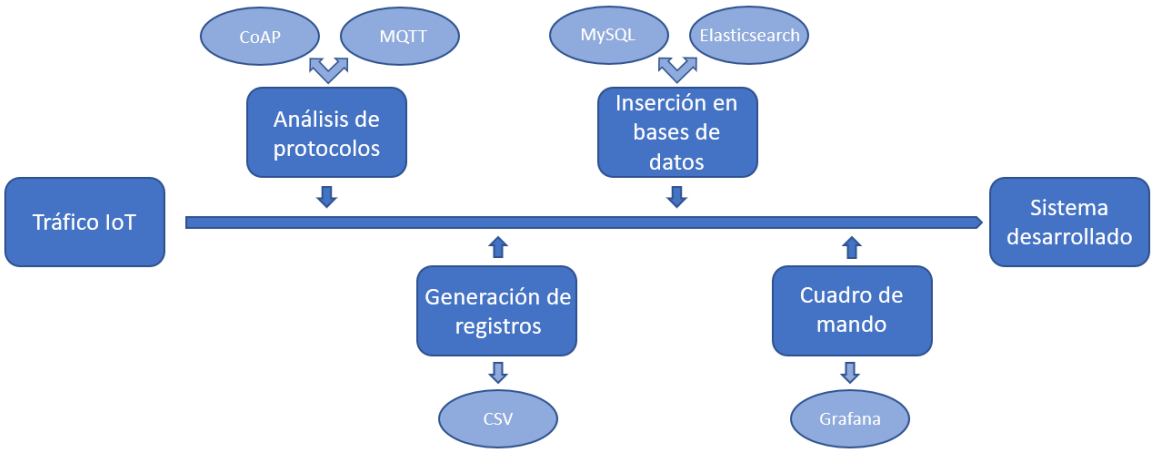

Figura 4.1: Diagrama del desarrollo realizado.

4.2. Análisis de los protocolos

Se comienza el desarrollo del sistema de monitorización de los protocolos MQTT y CoAP, con el análisis del tráfico transportado a través de estos protocolos. Para ello, se ha realizado un programa que se encarga de leer los paquetes y posteriormente procesar los datos mediante la extracción de los parámetros que transportan.

Este desarrollo se ha realizado haciendo uso del lenguaje de programación C. Esto es debido, a que este cuenta con una mayor rapidez a la hora de ejecutar este tipo de tareas, como son la lectura de archivos *pcap* o captura de tráfico en tiempo real, con respecto a otros lenguajes de programación, como puede ser Python. En cuanto al tipo de desarrollo seguido, se ha tomado como referencia un trabajo realizado con anterioridad [26] por el mismo autor que el de este presente documento.

Se parte de un archivo *pcap*, en el que se encuentra el tráfico capturado que se desea analizar. En este tráfico, si los paquetes son del protocolo MQTT o CoAP, se procesan los paquetes, y de descarta su análisis en caso contrario. Además, deben de usar la cabecera Ethernet. Este último requisito se impone debido a que el propósito de este trabajo no se centra en el análisis de la capa de enlace, sino de las de los protocolos mencionados, es decir, de la capa de aplicación. Como se menciona en la **Sección 6.2**, se propone el análisis de tráfico con otros niveles de enlace y red como trabajo futuro.

Para la lectura de este archivo, dado que usamos C como lenguaje de programación, se decide usar la biblioteca *libpcap*, la cual proporciona las funciones necesarias para la lectura de estos ficheros *pcap*. Una de las funciones usadas es:

```
pcap_open_offline_with_tstamp_precision(const char *fname,
                u_int precision, char *errbuf)
```

Como se puede intuir por el nombre de la función, esta se encarga de abrir un archivo que se le pase como argumento en la primera posición (*fname*), siempre y cuando este sea de tipo *pcap*. En cuanto al resto de argumentos, el argumento de *precisión* indica el rigor con el que se tomarán las marcas de tiempo o *timestamps* de los paquetes, que pueden ser segundos, microsegundos o nanosegundos. En el caso del desarrollo realizado, se ha optado por usar una precisión de microsegundos, debido a que la naturaleza de las redes IoT implica que la unidad del segundo limite la precisión. Por último, la variable *errbuff* se encarga de almacenar los códigos de error, si es que se llegan a producir. Si todo ha sido correcto, la función devolverá un puntero a la posición del primer paquete leído (*pcap_t **), o NULL en caso contrario.

Una vez tenemos este puntero, procedemos a la lectura de los paquetes de forma individual. Para esto, hacemos uso de la siguiente función, también de la misma librería que la anterior mencionada:

```
pcap_next_ex(pcap_t *p, struct pcap_pkthdr **pkt_header,
                const u_char **pkt_data)
```

Con esta función, se irá leyendo el archivo, paquete a paquete, devolviendo 1 si todo ha salido bien, o -2 si, por el contrario, ya no hay más paquetes que leer y por lo tanto hemos terminado de analizar la traza. Por ello, incluiremos en nuestro programa un bucle *while* que se irá ejecutando hasta que esta función devuelva el -2, indicando la finalidad del documento, asegurándonos de esta manera que todos los paquetes son leídos.

En cuanto a las variables que usa esta función, la primera sería el puntero anteriormen-

te extraído con la función `pcap_open_offline_with_tstamp_precision`, el segundo es directamente un puntero ligado a una estructura generada para el paquete y, por último, un puntero hacia los datos del paquete en sí.

También destacar que se puede comprobar el tipo de enlace que usa el paquete. Para esto se puede hacer uso de la función `pcap_datalink(pcap_t *p)`, la cual queda implementada en el código para futuras versiones del programa.

Después del uso de estas funciones, se inicia el análisis del paquete, extrayendo los campos que lo conforman de manera individual. Se hacen uso de estructuras, las cuales irán conformando cada una de las capas que se quieren analizar. En estas estructuras, se incluyen los campos que forman la cabecera de la capa, indicando el número de bytes que ocupan en ella.

No obstante, dado que no todas las cabeceras de un mismo protocolo cuentan con los mismos parámetros, se requiere que, en ciertas ocasiones, para un mismo protocolo existan varias estructuras que conformen los parámetros que se quieren extraer en cada una de las situaciones que presente el protocolo. Por ejemplo, no contamos con los mismos parámetros en un mensaje de tipo CONNECT en MQTT que de tipo PUBLISH, como se ha visto en la *Sección 2.3*. Un ejemplo de estas estructuras lo podemos ver en la **Figura 4.2**, las cuales se describirán más adelante.

```
/*MQTT fixed header*/
struct sniff_mqtt {
    uint8_t control;    //Control Field. First 4 Packet Type, next Flags
    u_char packet_length;  //MSB used as a continuation flag

};

/*MQTT connect command variables*/
struct sniff_mqtt_connect_1{
    uint16_t protocol_name_length;
    char protocol_name[];

};
struct sniff_mqtt_connect_2{
    uint8_t version;
    uint8_t flags;
    uint16_t keep_alive;
    uint16_t client_id_length;
    char client_id[];
};
struct sniff_mqtt_connect_3{
    uint16_t user_name_length;
    char user_name[];
};

/*MQTT Connect ACK*/
struct sniff_mqtt_ack{
    uint8_t reserved;
    uint8_t return_code;
};

/*MQTT Publish Message*/
struct sniff_mqtt_publish{
    uint16_t topic_length;
    char topic[];
};
```

Figura 4.2: Ejemplo de unas estructuras de MQTT.

Una vez que hemos creado todas las estructuras de las cabeceras que queremos analizar, procedemos a la extracción de los parámetros uno a uno. Como vamos a tratar con dos protocolos distintos, se explicarán de manera individual cada uno de ellos.

4.2.1. Análisis del protocolo MQTT

Como hemos visto en la **Figura 4.2**, se han generado varias estructuras para cada tipo de mensaje y cabecera. La primera que podemos ver en esta figura conforma la cabecera fija del paquete MQTT descrita en la **Sección 2.3**. Como se puede apreciar, al desconocer el valor real del campo longitud de paquete, ya que este dependerá de cada paquete, se establece con una constante de tipo *char* de manera que se puede ir asignando el valor a medida que se va calculando.

Esta última situación descrita es muy habitual en este tipo de protocolo, ocurre con otras variables como *Protocol Name*, *Client ID*, *User Name*, *Topic*, etc. Por esta razón se deben de generar numerosas estructuras para cada tipo de mensaje y a su vez, para un mismo mensaje otras tantas. En el ejemplo de la **Figura 4.2**, podemos ver como para un mensaje de tipo *CONNECT* poseemos 3 estructuras que conforman los parámetros que queremos extraer.

Comenzando con el análisis del paquete, se inicia extrayendo los datos de la cabecera IP, más concretamente las direcciones IP origen y destino, que serán de utilidad a la hora de la representación gráfica, como veremos en el **Apartado 4.4**. También, se leerá el campo que indica el protocolo que se usa, que en estos casos deberá ser TCP (con un valor de 6) y el tamaño total de la cabecera IP, de manera que se pueda saber donde comienza la siguiente cabecera.

Una vez terminado de extraer esos parámetros, se continua con la comprobación del protocolo, el cual, como se acaba de mencionar, deberá ser TCP. Si no fuera así, el paquete se descartaría y se continuaría leyendo el siguiente paquete.

De la cabecera TCP se extraerán el parámetro que indica su tamaño de cabecera, así como, los puertos origen y destino. Estos dos últimos valores son de gran importancia, ya que con ellos podemos conocer si el protocolo que viene a continuación es MQTT. Para que así sea, uno de estos deberá ser tener el valor 1883, el cual conforma el puerto del *broker* MQTT. De nuevo, si ninguno de estos puertos coincide con este valor, se descarta el análisis del resto del paquete.

Después de comprobar que se trata de un paquete con protocolo MQTT, procedemos a extraer los parámetros que consideramos relevantes a la hora de servir como información, de cara a un análisis de la red. No obstante, cabe recalcar que inicialmente se extrajeron más campos que los que procedemos a citar, sin embargo, se desechó su uso, dado que no se encontraron situaciones donde otorgaran funcionalidad. Hablamos por ejemplo del campo *Protocol Name*, el cual indicaba, en versiones antiguas, la versión del protocolo usado mediante un nombre identificativo para cada versión. Este campo se ha desechado su uso en la última versión de MQTT (5.0) ya que se posee un parámetro redundante con el campo *Protocol Version*. Otro campo no usado es la variable *Password*, que indicaba la contraseña usada para la transmisión. No se decidió usar ya que no aportaba ningún tipo de valor a la monitorización del sistema y en ocasiones se transmite de manera cifrada.

Centrándonos en los campos a destacar, podemos hablar del *Message type*, el cual

indica que tipo de mensaje se está produciendo y también nos indica el tipo de estructura que deberemos usar. Otro campo es la longitud del paquete (*Packet Length*). Este campo lo usaremos para conocer si, tras la cabecera MQTT que estamos analizando, se encuentran más cabeceras como esta o, si por el contrario, se ha llegado al final del paquete. Esto es realmente útil para aquellos paquetes que transportan múltiples cabeceras MQTT, como puede ocurrir en una transmisión donde está ocurriendo un ataque por inundación, como vemos en la **Figura 4.3**.

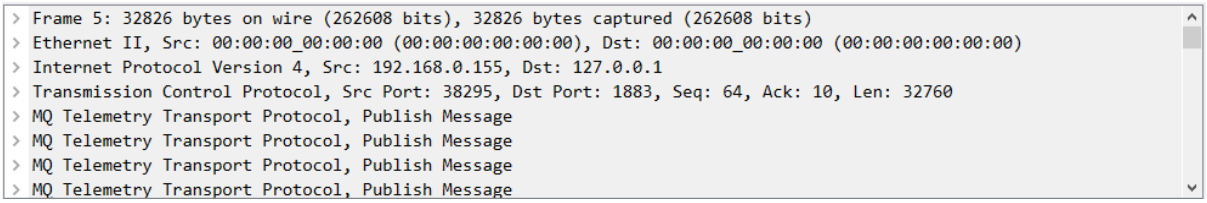

Figura 4.3: Paquete con múltiples cabeceras MQTT.

Prosiguiendo con la extracción de los campos, nos centramos ahora en aquellos que son específicos al tipo de mensaje enviado. Como hemos dicho en numerosas ocasiones, solo se muestran los campos que se han considerado relevantes para este proyecto. No obstante, estos pueden ser ampliados en un futuro si así se desea.

En el caso de los mensajes de tipo CONNECT, extraeremos los campos de *Protocol Version*, indicando la versión del protocolo, *User name Flag*, con el cual veremos si el paquete cuenta con identificación por nombre de usuario, si es así, se extraerá a mayores el parámetro *User name*. Y por último, también leeremos el campo de *Client Id*.

En cuanto al caso de los mensajes de tipo CONNACK, se extraerá el campo de *Return Code*, indicando el estado de la transmisión, es decir, si es errónea o no la petición recibida previamente.

De los paquetes de tipo PUBLISH, nos quedaremos con la variable *Topic*.

Por otro lado, también se extrae de cada paquete el tiempo de llegada *Unix*[1]. Este tiempo de llegada, se usará como campo para la representación y también, mediante la comparación de flujos, es decir retransmisiones que forman parte de una misma comunicación (CONNECT al *broker*, *broker* responde con un CONNACK), se generará el campo de tiempo de respuesta. Es decir, segundos de diferencia que hay entre paquetes de un mismo flujo.

Todos estos parámetros se irán almacenando en una tabla *hash*, de la que posteriormente se irá escribiendo todos estos parámetros mencionados en un archivo *CSV*, el cual se cargará en la base de datos, como explicamos en la **Sección 4.3**. Cabe mencionar que este archivo no se encuentra ordenado temporalmente ya que no es necesario. Estos serán ordenados según el tiempo de llegada por la herramienta de visualización que veremos en la **Sección 4.4**. Un ejemplo de este archivo *CSV* lo podemos ver en la **Figura 4.4**.

[1]Tiempo transcurrido, en segundos, desde la medianoche UTC del 1 de enero de 1970.

id	Init_time	Packet_type	Sequence_number	IP_src	IP_dst	Port_src	Port_dst	Protocol_Name	Client_Id
49071	2020-02-14 11:26:31	CONNECT	12270	10.0.0.16	192.168.1.7	32774	1883	MQIsdp	mosqpub/8473-box
49074	2020-02-14 11:26:31	CONNACK	12270	192.168.1.7	10.0.0.16	1883	32774		
49079	2020-02-14 11:26:31	PUBLISH	12270	10.0.0.16	192.168.1.7	32774	1883		
49081	2020-02-14 11:26:31	Disconnect_Req	12270	10.0.0.16	192.168.1.7	32774	1883		

User_Name	Return_Code	Topic	Answer_delay	Bytes	Number_Publish	QoS_publish	Version
eqS33			0	115	0	0	MQTT v3.1
	0		0.000236	70	0	0	
		sensor/sense	0.001697	87	1	0	
			0.001701	68	0	0	

Figura 4.4: Ejemplo del archivo CSV generado tras el análisis de paquetes MQTT.

4.2.2. Análisis del protocolo CoAP

Dejando a un lado el protocolo MQTT, nos centramos ahora en explicar las características del análisis de los paquetes del protocolo CoAP.

Al igual que para MQTT, en CoAP se han generado las estructuras necesarias para poder monitorizar el protocolo. Un ejemplo de las estructuras generadas es el de la **Figura 4.5**.

```
/*COAP header*/
struct sniff_coap{
    uint8_t control;    //version 2bits, type 2bits, token length 4bits
    uint8_t code;
    uint16_t message_id;
};

struct sniff_coap_options1{
    uint8_t option_delta_length;
    char uri[];
};

struct sniff_coap_options2{
    uint8_t option_delta_length;
    char uri[];
};

struct sniff_coap_endoptions{
    uint8_t end_options_marker;
};
```

Figura 4.5: Ejemplo de estructuras CoAP.

En cuanto al procedimiento seguido por el programa, es exáctamente el mismo que en MQTT. Empezando con la cabecera IP, con la extracción de las direcciones IP y protocolo. En este caso, a diferencia que con MQTT, CoAP funciona sobre el protocolo UDP. Por lo tanto, se espera que la variable protocolo sea igual al número 17. Si es así, se prosigue con en análisis extrayendo los puertos UDP. Dado que CoAP trabaja sobre el puerto 5683 en su servidor, uno de estos debe de coincidir para establecer que estamos usando este protocolo.

Una vez determinado que estamos ante un paquete de tipo CoAP, nos centramos en extraer los parámetros que consideramos relevantes. Al igual que con MQTT, optamos por leer la versión del protocolo que se esta usando y, de la misma manera, el tipo de mensaje que se está transmitiendo (*CONFIRMABLE, NON-CONFIRMABLE, AKNOWLEDGEMENT y RESET*).

También, se analiza la variable *Code*, que indica el tipo de petición o respuesta que se está realizando, como vimos en la **Tabla 2.5** de la **Sección 2.4**. Y el parámetro *Message ID*, para detectar duplicados.

Además, se extraen las variables de *Uri Path* y *Uri Host* de aquellos paquetes que la contengan. Dado que no son parámetros siempre presentes en todo mensaje CoAP, primero comprobamos que tipo de *Code* se está enviando en la transmisión, ya que no todos contienen el parámetro de opciones. Una vez filtrado, extraemos los parámetros *Option Delta* y *Option Length*, los cuales indican si contamos con una *Uri Path* o *Uri Host*, y su longitud respectivamente.

Por último, al igual que para MQTT, obtendremos el tiempo de llegada de los paquetes en UNIX. También se hará el cálculo del tiempo de respuesta de paquetes de un mismo flujo.

De nuevo, todos estos parámetros se irán almacenando en una tabla *hash*, la cual se recorrerá al final de la lectura de todos los paquetes, para ir imprimiendo todos los registros generados en un archivo *CSV*, como el que podemos ver en la **Figura 4.6**.

Init_time	Message_id	Packet_type	IP_src	IP_dst	Port_src	Port_dst	Code	RTT	Bytes	Version	Uri_host	Uri_path
2021-02-07 15:36:47	42214	CONFIRMABLE	192.168.0.151	134.102.218.18	50779	5683	PUT	0	70	1	coap.me/	/sink
2021-02-07 15:36:47	42214	AKNOWLEDGEMENT	134.102.218.18	192.168.0.151	5683	50779	2.04_Charged	0.043676	56	1		

Figura 4.6: Ejemplo del archivo CSV generado tras el análisis de paquetes CoAP.

4.3. Bases de datos

Una vez que se han generado los registros y se han almacenado en los archivos *CSV*, se procede a la carga de estos mismos hacia unas bases de datos. Estas bases de datos, serán a las que Grafana accederá para poder así representar los datos gráficamente.

Como se explicó en la **Sección 2.7**, se ha hecho uso de dos tipos distintos de bases de datos. Una que se basa en SQL, como es MySQL, y otra que no, como Elasticsearch. Se ha decidido hacer uso de ambos métodos para poder así ver cual de los dos tipos de bases de datos se adecúa más según los casos de uso. Esta comparación se expondrá más adelante en este documento, más concretamente en el **Capítulo 5**.

4.3.1. MySQL

MySQL es un sistema que gestiona bases de datos mediante el lenguaje SQL. Lo primero que se deberá hacer para poder usarlo es instalarlo. Para ello, partiendo en nuestro caso de una distribución Linux, más concretamente Ubuntu, se necesita el siguiente comando [27]:

```
sudo apt install mysql-server
```

También, aunque es opcional, se puede añadir seguridad al sistema de manera que requiera de contraseñas para su uso, utilizando el siguiente comando:

```
sudo mysql_secure_installation
```

Una vez que tenemos el servidor de MySQL configurado, si hemos seguido los pasos mostrados y hemos añadido seguridad deberemos acceder a la *shell* de MySQL de la siguiente manera:

```
mysql -u root -p
```

Cuando ya tenemos configurado el servicio de MySQL y hemos podido acceder a él, ya se puede iniciar la carga de datos. Se comienza generando una base de datos, formada por tablas, que son las que contendrán la información inyectada desde los archivos de registros generados anteriormente. Estas bases de datos que se generan no necesitan estar relacionadas, ya que se generará una para el caso de CoAP y otra para MQTT. Con esto llegamos a la conclusión que una base de datos no relacional también funcionaría adecuadamente en este proyecto.

Por lo tanto, se inicia con la creación de la base de datos. Esto se hace mediante el siguiente comando:

```
CREATE DATABASE <nombre de la base de datos>
```

En el caso particular de este proyecto, la hemos nombrado como *IoTDataBase*.

Proseguimos generando las tablas. Estas, deben de contar con las columnas que deseemos insertar, así como el tipo de dato de cada columna. El primer parámetro que se le indicará, es el nombre que se le va a otorgar a la tabla, seguido de un paréntesis donde se introducirán los parámetros con los que debe de contar esta.

En este desarrollo se han realizado 2 tablas distintas, una para MQTT y otra para CoAP. La semántica usada para la generación de las tablas es la siguiente:

Para MQTT:

```
CREATE TABLE MQTT (
   id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
   Init_time DATETIME,
   Packet_type VARCHAR(255),
   Sequence_number INT,
   IP_src VARCHAR(255),
   IP_dst VARCHAR(255),
   Port_src INT,
   Port_dst INT,
   Protocol_Name VARCHAR(255),
   Client_Id VARCHAR(255),
   User_Name VARCHAR(255),
   Return_Code INT,
   Topic VARCHAR(255),
   Answer_delay DOUBLE,
   Bytes INT,
   Number_Publish INT,
   QoS_publish INT,
   Version VARCHAR(255) );
```

Para CoAP:

```
CREATE TABLE CoAP (
   id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
   Init_time DATETIME,
   Message_id INT,
   Packet_type VARCHAR(255),
   IP_src VARCHAR(255),
   IP_dst VARCHAR(255),
   Port_src INT,
   Port_dst INT,
   Code VARCHAR(255),
   RTT DOUBLE,
   Bytes INT,
   Version INT,
   Uri_host VARCHAR(255),
   Uri_path VARCHAR(255) );
```

Como podemos ver, se han puesto ejemplos de distintos tipos de variables utilizadas. Una de tipo INT, es decir un número entero, otra de tipo VARCHAR(255), la cual indica una linea de texto y, por último DATETIME, que se encarga de transformar un tiempo UNIX a la fecha que corresponde, siguiendo el siguiente formato 'YYYY-MM-DD hh:mm:ss' (año-mes-día hora:minutos:segundos). Esta resolución en segundos viene dada debido a que la herramienta de visualización que usaremos no permite la representación de elementos en tiempos inferiores al segundo, por lo que la resolución de los microsegundos, mencionada previamente, queda relegada únicamente al cálculo del tiempo de respuesta.

Finalmente, procedemos a la carga de los datos desde el *CSV* a la tabla generada. Para esto, se necesita poner la variable *local infile* a *true*, permitiendo la carga de datos desde archivos locales. Esto se hace con el siguiente comando:

```
set global local_infile=true
```

Tras ello, se cargan los datos. Cabe destacar que se debe de indicar el tipo de delimitador usado en el archivo y si deseamos que se salte la carga de la primera linea del fichero. En el caso de este proyecto, se ha usado el delimitador **;** y se anula la carga de la primera linea del fichero la cual está formada por los nombres de las columnas generadas. Un ejemplo del comando usado es el siguiente:

```
LOAD DATA LOCAL INFILE 'regisrtroMQTT(COAP).csv'
  INTO TABLE <nombre de la tabla (MQTT o CoAP)>
FIELDS TERMINATED BY ';' LINES TERMINATED BY '\n' IGNORE 1 ROWS
```

Tras estos pasos, la tabla generada se encontrará rellenada con los registros que poseíamos en el *CSV*, permitiendo así la posterior consulta por parte de Grafana.

4.3.2. Elasticsearch

Como se ha introducido previamente, dado la sencillez de las bases de datos planteadas permite que un modelo de base de datos no relacional funcione de manera adecuada. Por ese motivo introducimos la alternativa a MySQL con Elasticsearch. Esta es una base de datos orientada a documentos de tipo *JSON*, a los cuales se les puede consultar, crear, actualizar o eliminar datos sobre ellos[28].

Su instalación es muy sencilla. En la página web de elastic[2] podemos encontrar los archivos comprimidos necesarios para su funcionamiento, para cada uno de los distintos sistemas operativos donde puede funcionar.

Una vez se ha instalado y se han descomprimido los archivos, se puede arrancar el servidor de Elasticsearch ejecutando el siguiente comando dentro de la carpeta:

```
./bin/elasticsearch
```

Ya finalizada la instalación y arrancado el servidor, podemos comenzar a inyectar los datos a Elasticsearch. Para ello, se ha usado una herramienta desarrollada por el mismo grupo que Elastic, hablamos de Logstash.

Logstash

Logstash se usa para agregar, procesar los datos y enviarlos a la base de datos de Elasticsearch. Una de las principales características de Logstash, es que permite cargar los datos de múltiples fuentes de manera simultanea y está constantemente cargando las actualizaciones se se hagan en los ficheros de partida, de manera que se incluyan en la base de datos de forma instantánea.

En este desarrollo, la utilidad dada a Logstash es de *pipeline* entre el archivo *CSV* de registros del que partimos, a la base de datos de Elasticsearch.

Para esto, se ha configurado un archivo, con extensión *.conf*, en el que se indica el proceso a realizar. Un ejemplo de este archivo lo podemos ver en la **Figura 4.7**.

```
input {
    file {
        path => "/home/ubuntu/TFM/registerMQTT.csv"
        start_position => "beginning"
        sincedb_path => "/dev/null"

    }
}
filter {
    csv {
        separator => ";"
        columns => ["id","Init_time","Packet_type","Sequence_number","IP_src",
                    "IP_dst","Port_src","Port_dst","Protocol_Name","Client_Id",
                    "User_Name", "Message_identifier","Return_Code","Topic",
                    "RTT","Bytes","Number_Publish","QoS_publish"]
        skip_header => "true"
    }
    if [id] == "id" { drop {} }
    mutate {
        remove_field => ["message", "@version", "hots", "path"]
        convert=> {
            "Bytes" => "integer"
            "RTT" => "float"
            "Port_src" => "integer"
            "Port_dst" => "integer"
            "Message_identifier" => "integer"
            "Return_Code" => "integer"
            "Number_Publish" => "integer"
            "QoS_publish" => "integer"
        }
    }
    date {
        match => [ "Init_time", "yyyy-MM-dd HH:mm:ss"]
        target => "@timestamp"
    }
}
output {
    elasticsearch {
        hosts => "http://localhost:9200"
        index => "mqtt_register"
    }
    stdout{}
}
```

Figura 4.7: Ejemplo de archivo de configuración de Logstash para MQTT.

[2] `https://www.elastic.co/es/downloads/elasticsearch`

En este fichero mostrado, del protocolo MQTT, se pueden distinguir tres zonas delimitadas con los nombres *input*, *filter* y *output*. En la zona de *input*, lo principal a destacar es la variable *path*, que deberá estar igualada a la localización del fichero que queremos inyectar en la base de datos.

Por otro lado, en la zona de *filter*, podemos distinguir varias opciones seleccionadas:

- Un apartado con nombre *csv*, que está indicando las características del archivo que vamos a tratar. En este caso, como su propio nombre indica, es un archivo *csv*, usando como separador de registros **;** e indicando las columnas correspondientes.
- Un apartado con nombre *mutate*, que nos indica, en este caso, ciertas columnas que queremos omitir, ya que estas vienen prefijadas en Logstash y, también, un conversor de variables, de manera que los números sean tratados como tal, debido a que todos los registros son procesados como texto si no se indica lo contrario.
- Por último, destacar el apartado *date*, indicando el formato en el que se quiere la fecha y asignándole a la variable *timestamp*, que es la que usa Elasticsearch para los tiempos.

Finalmente, nos encontramos con la zona de *output*, donde se índica el puerto e IP donde Elasticsearch se encuentre funcionando, así como, el índice donde se almacenarán los registros.

En cuanto al fichero de CoAP, lo podemos ver en la **Figura 4.8** y como se puede apreciar sigue exáctamente la misma estructura que para MQTT.

```
input {
    file {
        path => "/home/ubuntu/TFM/registerCOAP.csv"
        start_position => "beginning"
        sincedb_path => "/dev/null"
    }
}
filter {
    csv {
        separator => ";"
        columns => ["Init_time","Message_id","Packet_type","IP_src",
                    "IP_dst","Port_src","Port_dst","Code","RTT",
                    "Bytes","Version","Uri_host","Uri_path"]
        skip_header => "true"
    }
    if [id] == "Init_time" { drop {} }
    mutate {
        remove_field => ["message", "@version", "hots", "path"]
        convert=> {
            "Bytes" => "integer"
            "RTT" => "float"
            "Port_src" => "integer"
            "Port_dst" => "integer"
            "Message_id" => "integer"
            "Version" => "integer"
        }
    }
    date {
        match => [ "Init_time", "yyyy-MM-dd HH:mm:ss"]
        target => "@timestamp"
    }


}
output {
    elasticsearch {
        hosts => "http://localhost:9200"
        index => "coap_register"
    }
    stdout{}
}
```

Figura 4.8: Ejemplo de archivo de configuración de Logstash para CoAP.

Una vez tenemos configurado el fichero, únicamente debemos lanzar el siguiente comando(siempre y cuando el servidor de Elastisearch esté funcionando):

```
./logstash -f logstash.conf
```

Y, gracias a que en el fichero de configuración se ha optado por escribir por pantalla todos los registros, se podrá ver cuando la carga de datos ha sido completada.

4.4. Visualización de datos

Una vez que tenemos las bases de datos cargadas con los registros, podemos comenzar a operar con ellas. Haremos uso de Grafana, que como se habló en la **Sección 2.7**, se trata de una herramienta usada para analizar registros, monitorización en tiempo real, monitorización de aplicaciones, etc.

Se decide el uso de esta herramienta, en comparación con otras disponibles como puede ser Kibana[3], debido a que es la más usada de las que existen de código abierto y, a mayores, la que permite mayor flexibilidad a la hora de usar distintos tipos de fuentes de datos. Ya que Kibana, su mayor competidor, se limita al uso de Elasticsearch, mientras que Grafana, aparte de poder usar Elasticsearch, también permite usar otras bases de datos como InfluxDB, Prometheus, MySQL, entre otras. Además, estas bases de datos pueden usarse de manera simultánea en un mismo panel, aumentando la flexibilidad de cara al desarrollador.

Para instalarlo, se puede hacer mediante uso de la terminal o, al igual que Elasticsearch, descargando los binarios comprimidos en un archivo zip. De manera que con únicamente descomprimirlo ya podremos hacer uso de la herramienta.

Para arrancar el servidor, bastará con ejecutar el siguiente comando(dentro de la carpeta con los binarios):

```
./bin/grafana-server
```

En cuanto a la interfaz de usuario, se accede a ella mediante el navegador, indicando la dirección IP donde esté corriendo el servidor y el puerto 3000.

Cuando accedemos a la interfaz de Grafana, lo primero que se ha realizado es enlazar las bases de datos con las que contábamos. Esto se realiza accediendo al apartado de configuración de Grafana y luego, seleccionando *Data Sources*, podremos añadirlas. Para el caso de MySQL, debido a que no se encuentra directamente como *plugin* dentro de la configuración básica de Grafana, es necesario añadirlo, no obstante en la página web de Grafana podemos añadirlo con facilidad[4].

Para añadir estas bases de datos, en la configuración de Grafana es necesario indicar la dirección y puerto donde se encuentran alojados en sus respectivos servidores, así como el nombre de la base de datos para MySQL y el nombre del registro para Elastisearch.

Una vez tenemos enlazadas las bases de datos, se comienza con la creación de los paneles (*Dashboards*) donde se realiza la representación de los datos. Ejemplo de ello, podemos verlo en las **Figuras 4.9, 4.11 y 4.13**.

[3] `https://www.elastic.co/es/kibana`

[4] `https://grafana.com/grafana/plugins?type=datasource&utm_source=grafana_add_ds`

A continuación, nos centramos en explicar las decisiones llevadas a cabo para la representación gráfica. Hablaremos de cuales son los datos mostrados y el porqué de estos.También se comentará el tipo de petición realizada para poder hacer estas representaciones, se mostrarán las aquellas más significativas. Los ejemplos mostrados proceden de una captura de tráfico MQTT de unos 20 minutos de duración y 50000 paquetes.

Comenzamos con la **Figura 4.9**, en ella podemos apreciar varias tablas, una gráfica y filtros de selección en el encabezado de la página.

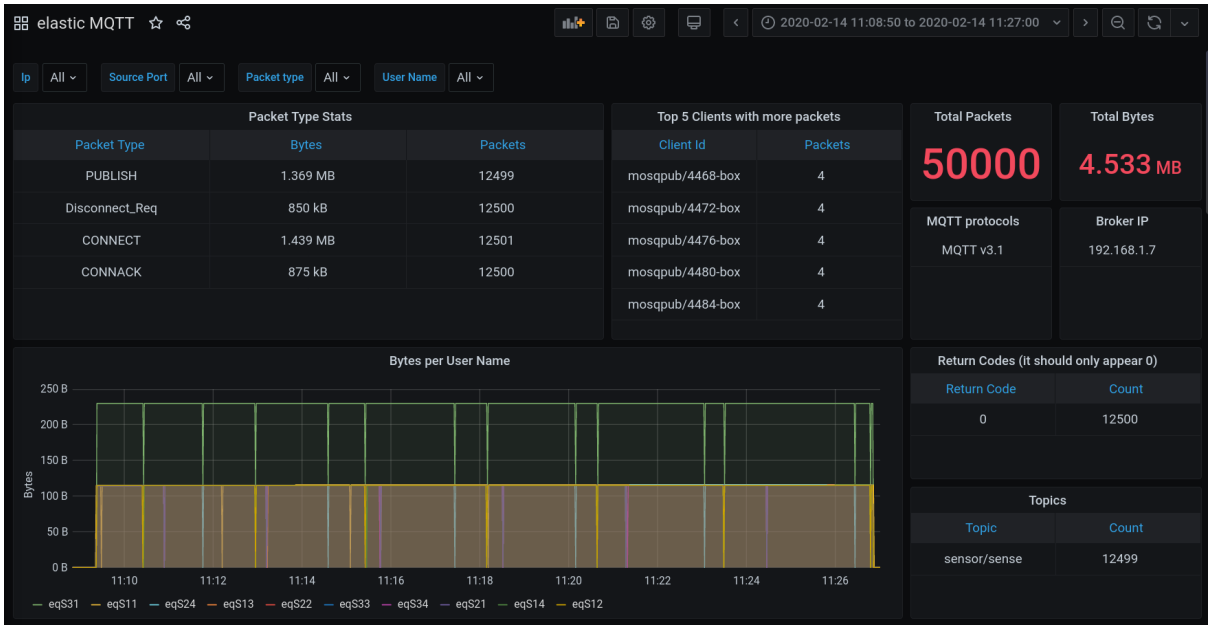

Figura 4.9: Dashboard de ejemplo. Protocolo MQTT con Elasticsearch como base de datos.

Empezando por la tabla con título *Packet Type Stats*, en esta se muestra el número de paquetes que se han transmitido, así como, los bytes de estos, por cada tipo de mensaje. Esta tabla aporta una información muy relevante al gestor de la red IoT, ya que puede localizar algún tipo de irregularidad, si por ejemplo, muchos dispositivos se están intentando conectar de forma inesperada.

La formación de esta tabla la podemos ver en la **Figura 4.10**. Podemos apreciar el primer campo, con nombre *Query*, en el cual igualamos una de las variables a representar, (*Packet type*) al valor que toma el filtro que hemos puesto para esta variable. De esta manera lo parámetros a representar podrán variar según el filtro seleccionado. En cuanto a los campos *Metric*, en ellos hemos realizado dos operaciones. La primera suma el parámetros byte y la segunda cuenta el número de veces que se repite, en este caso, el campo *Packet type* por el que se agrupan, como vemos en la última fila.

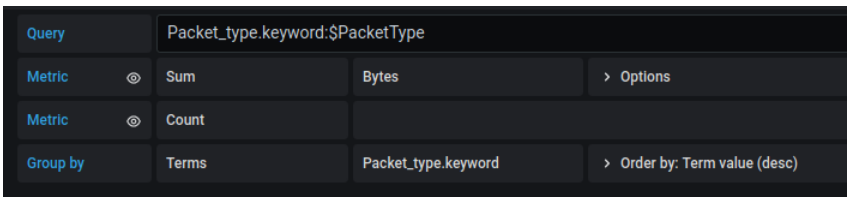

Figura 4.10: Ejemplo de la formación de la tabla de estadísticas de *Packet Type*.

Volviendo a la descripción del tablero, a su derecha, podemos ver otra tabla indicando los 5 clientes con mayor tráfico de paquetes que, al igual que el caso anterior, puede determinar si hay alguna irregularidad con algún cliente en particular.

Se prosigue con una serie de datos meramente informativos, como pueden ser el número total de paquetes, bytes, las versiones del protocolo usadas, la dirección IP del *broker* utilizada, el número de *Return Codes* detectados, de manera que se pueda verificar que todos son correctos. Si se encuentra alguno distinto de 0 es signo de algún error en la transmisión. También se muestran los distintos *topics* publicados.

Por último, vemos la gráfica de número de bytes por *User Name* a lo largo del tiempo. El *User Name* se usa como medio de autenticación para el *broker*, y una variación muy pronunciada puede indicar una anomalía en la red o incluso un ataque.

Cabe también destacar que, en la parte superior, se presentan una serie de variables que permiten filtrar casos concretos de IP, por ejemplo, entre otros posibles filtros disponibles.

En cuanto a la **Figura 4.11**, podemos apreciar 6 tipos de gráficas distintas.

Figura 4.11: Continuación del dashboard de ejemplo. Protocolo MQTT con Elasticsearch como base de datos.

Comenzando por las dos de la parte superior, a la izquierda tenemos la relación número de paquetes/bytes a lo largo de la transmisión. El motivo de esta representación, está ligada hacia la detección de anomalías. Se espera que esta relación sea más o menos constante. No obstante, si detectamos zonas donde la acumulación de bytes es muy elevada frente a los paquetes, esto indica que los paquetes transmitidos contienen mucha más información de lo normal, lo que puede significar un intento de sobrecarga del servidor.

A su derecha, se puede apreciar la representación de cantidad de bytes por cada tipo de mensaje transmitido a lo largo de la captura. Esto, sirve de complemento a la primera tabla comentada, ya que se puede localizar, poniendo un ejemplo, el instante de tiempo donde se ha podido generar una inestabilidad detectada.

En cuanto a la fila central se puede apreciar que se ha centrado en la representación de los tiempos que se indican a continuación.

Por un lado, tenemos a la izquierda el tiempo máximo de respuesta detectado por cada dirección IP. Con esto, nos asegurarnos si algún dispositivo puede estar dando algún tipo de problema que le impida responder en el tiempo esperado, ya sea por falta de energía causada por una descarga de la pila que pueda usar, o simplemente por un embotellamiento en su trasmisión.

Por el otro lado, a la derecha, se representan los tiempos de respuesta, con los percentiles 95 y 99, de cada IP a lo largo del tiempo. Con esto, se puede identificar si los tiempos de respuesta están siendo los esperados o no. Cabe recalcar, que este cálculo del percentil, únicamente se puede hacer directamente desde Grafana con Elasticsearch. Para el caso de MySQL se necesitaría hacer el cálculo de los mismos mediante una consulta específica que haga el cálculo.

En la **Figura 4.12** podemos ver los parámetros requeridos en Grafana para la realización de esta gráfica. De nuevo, se establece en el parámetro *Query* el filtro que usamos. En este caso, igualamos la variable *Ip src* al filtro establecido par las Ips. En cuanto al campo *Metric*, se selecciona la operación que queremos realizar, en este caso percentil, seguido de la variable a operar junto a los tipos de percentiles deseados (95 y 99). Por último, lo agruparemos por el top 10 de Ips y se representará a lo largo del tiempo gracias al parámetro *timestamp*. El intervalo de este último lo dejaremos en automático, pero podrá ser modificado si se quiere una precisión de segundos concreta.

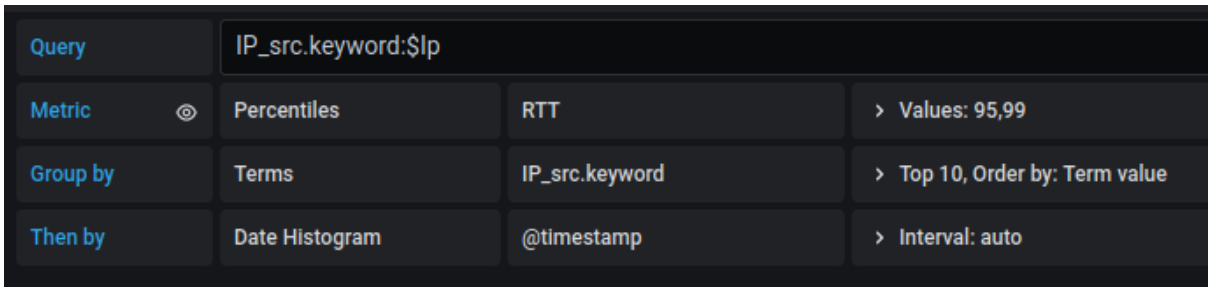

Figura 4.12: Ejemplo de formación de la gráfica de tiempos de respuesta con percentiles.

Volviendo al tablero, en la ultima fila se representa el tráfico, en bytes, transmitido por el *broker* y por los clientes. Con estas representaciones, podemos obtener información del estado de la transmisión según cada dirección IP, y detectar, por ejemplo, si el comportamiento de un dispositivo es el esperado o por el contrario está cometiendo algún tipo de irregularidad.

Terminando con la última imagen del tablero creado, **Figura 4.13**, en esta podemos ver una serie de gráficas y datos informativos.

Empezando por estos últimos, vemos como se marcan el número de paquetes y el total de bytes que se transmiten desde el *broker* y también los enviados hacia el *broker*. Estos datos, sirven de apoyo a las gráficas inferiores, las cuales indican datos similares, pero filtrándolos por tipo de paquete o de IP. Con estas representaciones se quiere poder mostrar el flujo de datos generado en las distintas direcciones, de manera que se pueda identificar posibles errores y localizarlos del lado del *broker* o el lado contrario si este fuera el caso.

Figura 4.13: Continuación del dashboard de ejemplo. Protocolo MQTT con Elasticsearch como base de datos.

En la **Figura 4.14**, se muestra la formación de la gráfica de bytes enviados por cada IP. La principal diferencia para la formación de esta gráfica, frente a la vista en la **Figura 4.12**, se encuentra en el campo *Metric*. En este caso se selecciona la operación de suma para la variable de los bytes. De esta manera, gracias a la agrupación por Ip y la representación en el eje X del tiempo, se puede ver el número de bytes totales que hay en cada segundo por cada Ip.

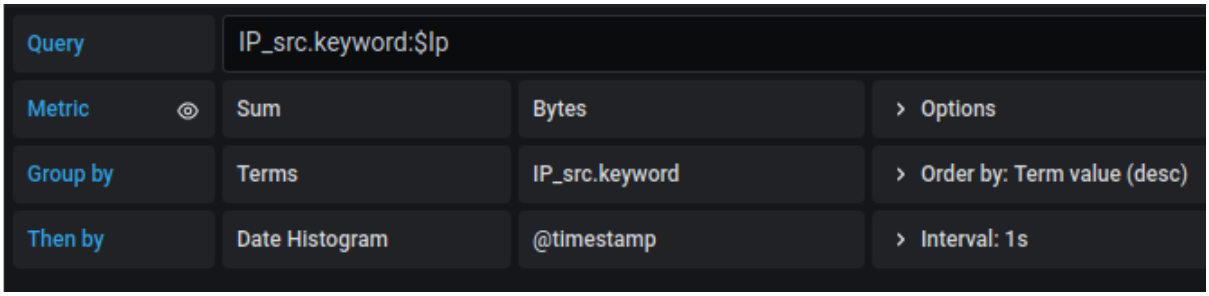

Figura 4.14: Ejemplo de formación de la gráfica de bytes enviados por Ip.

Hasta ahora las imágenes mostradas procedían de un tablero que usaba MQTT como protocolo. Para el caso de CoAP, el tablero cuenta con ciertas diferencias, aunque en esencia es muy similar. Por ejemplo, no cuenta con la representación de Bytes según *User Name* ya que este protocolo no cuenta con este parámetro, al igual que los *Topics* y *Return code*. Por el contrario, cuenta con ciertos paneles propios de este protocolo, como pueden ser el *Uri Host* y *Uri Paths* de la **Figura 4.15** o el parámetro *Code* de la **Figura 4.16**.

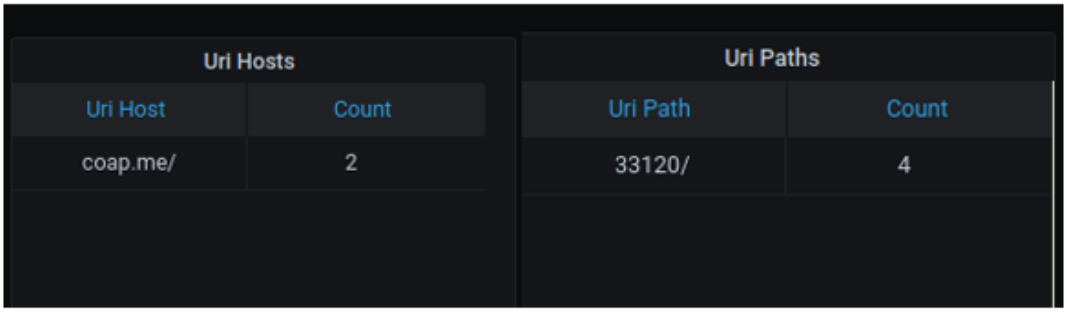

Figura 4.15: Ejemplo de tabla de *Uri Host* y *Uri Paths* para el tablero de CoAP.

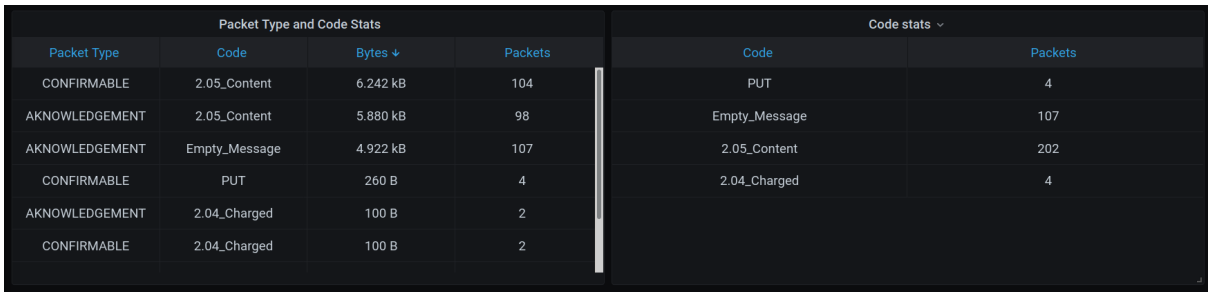

Figura 4.16: Ejemplo de tabla de *Code* para el tablero de CoAP.

Por último. debemos mencionar que se han implementado alertas para la detección de anomalías. Estas alertas, se deben situar conforme a la "normalidad" esperada en cada tipo de gráfico. Por lo tanto, se necesitan fijarse conociendo los valores medios aproximados de la red que se esté analizando. Con estas alarmas, se han podido detectar ataques por inundación.

4.5. Conclusión

A lo largo de este capítulo se han mostrado el desarrollo y diseño del sistema de monitorización llevado a cabo para la realización de este proyecto. Empezando por el análisis del tráfico capturado de MQTT o de CoAP, de los cuales se extraían los parámetros que se han considerado relevantes para posteriormente graficarlos.

Una vez generados y extraídos estos parámetros, los cuales se han almacenado en un archivo *CSV*, estos han sido puestos en las bases de datos probadas, MySQL y Elasticsearch. Estas bases de datos, serán consultadas por la herramienta de visualización Grafana, mediante la cual se han generado gráficos, tablas y registros que permiten al usuario conocer el estado de la red

No obstante, es necesario poder validar todo lo desarrollado hasta ahora, de cara a comprobar el correcto funcionamiento del sistema de monitorización de redes IoT. En el siguiente capítulo, se comentarán todas las validaciones llevadas a cabo en este Trabajo Fin de Máster.

5

Validación y pruebas

5.1. Introducción

En este capítulo, se comentarán todas las validaciones y test que se han realizado de cara a probar la funcionalidad y prestaciones del sistema.

Para ello, se explicará las distintas pruebas realizadas, así como, el funcionamiento de herramientas usadas para la ejecución de las mismas. Con esto, lo que se quiere lograr es demostrar la validez del desarrollo realizado y las decisiones tomadas.

La metodología seguida para la validación se puede ver en la **Figura 5.1**, donde se muestra un diagrama con todo el proceso, en orden, seguido a lo largo del proyecto para la validación del sistema desarrollado.

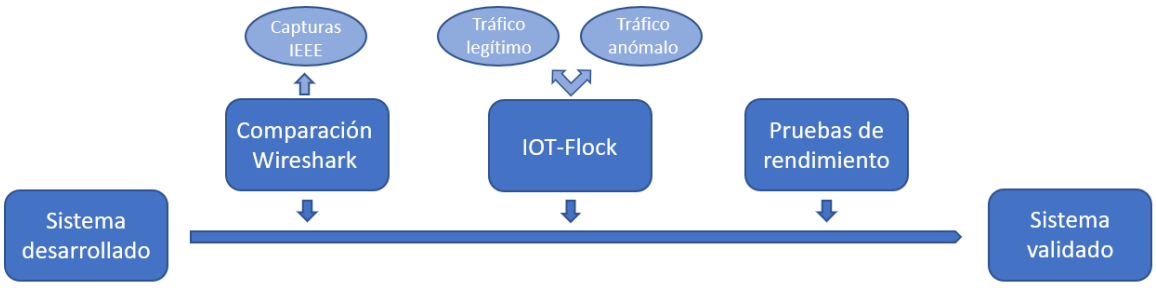

Figura 5.1: Diagrama de validación seguido en el proyecto.

5.2. Wireshark

A lo largo de todo el desarrollo, se presenta la necesidad de contar con herramientas que sirvan para contrastar si los resultados obtenidos son los adecuados. Una de las herramientas que se ha utilizado como forma de validación durante todo el proyecto es Wireshark.

Wireshark es una herramienta que permite la interceptación y análisis del tráfico de una red, transformando los paquetes capturados a un formato legible según el tipo de protocolo que se trate. En nuestro caso particular, se ha tratado de los protocolos MQTT y CoAP, como se ha ido viendo a lo largo de este documento, los cuales se encuentran disponibles dentro de la base de datos de Wireshark para su interpretación.

Al comienzo del proyecto, se utilizó esta herramienta como apoyo a la información obtenida en la etapa de *estado del arte* acerca de la estructura de los paquetes. Se utilizó haciendo uso de paquetes de ejemplo de los protocolos tratados, sirviendo de apoyo para desarrollo del analizador en C previamente mencionado.

Una vez que se desarrolló el analizador de protocolos, se necesitaba realizar pruebas antes de continuar con el proyecto, para verificar que los datos se estaban extrayendo correctamente. Se utilizó unos archivos *pcap* proporcionados por el IEEE [29] que contenían tráfico del protocolo MQTT.

La idea con ello, era comprobar los resultados obtenidos del analizador implementado frente a los que se extraían con Wireshark. Por ese motivo, se decide realizar una modificación al código desarrollado, de manera que se genere un segundo archivo *CSV* que plasme los resultados de una manera muy similar a los logrados con Wireshark. Más concretamente a los obtenidos en el apartado *Estadísticas/Conversaciones/IPv4* y *Estadísticas/Conversaciones/TCP* (TCP para el caso MQTT, UDP para CoAP).

Para el caso de CoAP se hizo el mismo procedimiento, pero se requirió usar herramientas para poder generar tráfico de este protocolo, ya que no se encontraron capturas de tráfico disponibles.

También se verificó, en entornos controlados con poca cantidad de paquetes de cada protocolo, pero usando distintos tipos de mensaje de manera que estuvieran representadas la mayor parte de situaciones, que los parámetros específicos de MQTT y CoAP extraídos estaban siendo correctamente analizados, sin errores. Con esto, se pudo detectar cierta cantidad de errores de programación que pudieron ser subsanados gracias a esta herramienta.

5.3. Generación de tráfico

Uno de los aspectos más importantes para validar este proyecto es el comprobar su funcionalidad, con situaciones reales o simuladas, de todo el proceso en su conjunto. Es por ello, que cobra vital importancia la capacidad de generar tráfico de los protocolos tratados en este trabajo.

Hablamos de vital importancia para la validación de este proyecto debido a la escasez de tráfico capturado que podemos encontrar en la red. A esta escasez, hay que sumarle la flexibilidad que supone el contar con herramientas que permitan, de una manera

controlada, generar distintas situaciones de tráfico.

A continuación hablaremos de las usadas en este trabajo. Podemos ver como se han descartado otras herramientas como Cooja o Netsim. La primera debido a que funciona sobre 6LoWPAN, lo que supone un gran impedimento para el desarrollo de este proyecto, debido a que implica una compresión de IPv6[1], lo que dificulta la tarea de análisis del tráfico al tener que añadir métodos de descompresión. Para el caso de Netsim, al ser de pago no se ha podido usar.

5.3.1. CoAP Shell

Como se comentó en la **Sección 2.6.2**, CoAP Shell es una herramienta que permite la interacción con servidores CoAP, pudiendo generar peticiones o consultar el estado de sensores que estén implantados en los servidores (pudiendo ser reales o simulados).

Para poder usarlo, haremos uso del proyecto publicado en Github [22], donde se nos indican las instrucciones para instalarlo. Podemos ver como, simplemente descargando la versión *build* que se nos indica, se puede comenzar su uso.

Una vez descargado, nos conectaremos al servidor CoAP que deseemos. En nuestro caso se ha usado `http://coap.me/`, pero también se han testeado otros. Para conectarnos se usa el siguiente comando:

```
connect coap://coap.me
```

Y con ello, tras hacer uso del comando `discover`, podemos conocer cuales son los recursos que se encuentran disponibles para su interacción, mediante las peticiones `get`, `put`, `post` y `delete`.

Gracias a esta herramienta, se pudo generar pequeñas cantidades de tráfico, que se usaron en las primeras etapas del desarrollo. No obstante, para la validez y la generación de trazas con mayor cantidad tráfico se uso la herramienta de la que hablaremos a continuación.

5.3.2. IoT-Flock

Dado que con CoAP Shell nos encontrábamos limitados únicamente a la generación de tráfico de tipo CoAP, se necesitaba también una herramienta que pudiese simular tráfico MQTT. Es aquí donde entra el proyecto de IoT-Flock.

Como se explicó en la **Sección 2.6.4**, se trata de una herramienta que permite la generación de tráfico, en tiempo real, de los protocolos IoT (MQTT y CoAP), mediante la simulación del funcionamiento de dispositivos IoT, como podría ser un sensor de luminosidad. A mayores, permite generar situaciones anómalas en la red, simulando ataques ya conocidos en estos protocolos.

Dado que el objetivo de esta herramienta es generar tráfico desde el punto de vista de un sensor, esta requiere de otras implementaciones para que hagan las funciones de *broker*, para el caso de MQTT, y de servidor para CoAP. Por este motivo, para el caso de MQTT, se hace uso de Mosquitto como *broker*, el cual se encuentra funcionando en

[1] `https://tools.ietf.org/html/rfc2460`

la dirección IP local de la máquina (127.0.0.1) y para el caso de CoAP, es necesario introducir la url donde se encuentre funcionando uno de los servidores CoAP ya tratados previamente.

Para hacer uso de esta herramienta, accederemos a la página de Github donde se encuentran alojados los archivos [30]. En ella, encontramos todos los pasos a seguir y los archivos a descargar, para poder poner en funcionamiento la herramienta. No obstante, durante este proceso surgieron problemas a la hora de poder hacer uso de la misma. Uno de los problemas detectados y del cual no se pudo subsanar, fue a la hora de usar la interfaz gráfica de usuario (*GUI*) de la herramienta, la cual no contaba con el funcionamiento esperado. Debido a esta situación, se decidió contactar con los responsables del proyecto, para así poder hacer uso de su desarrollo, los cuales otorgaron una maquina virtual, ya montada, con el sistema desarrollado funcional. Esto supuso de gran ayuda y se agradece a sus desarrolladores por la amabilidad y rapidez en el trato.

Una vez se encuentra la herramienta instalada, podemos hacer uso de ella. Para ello, accedemos a la carpeta donde se encuentran los binarios del sistema y desde ahí abrimos un terminal. Se comienza generando los casos de uso que queremos generar, es decir, el sistema de sensores que queremos simular, el tipo de mensaje que transmitirán estos sensores y la periodicidad del envío de los mensajes. Para comenzar, lanzamos el siguiente comando con el que accedemos a la interfaz de usuario:

```
./IoTFlock-GUI
```

Una vez lanzada la interfaz, podremos generar un nuevo caso de uso clicando en el botón `+Add New UseCase`. Con este, se creará un nuevo caso de uso donde podremos ir añadiendo los distintos dispositivos que queremos introducir a la simulación.

En la **Figura 5.2** podemos ver el procedimiento para la creación de un nuevo dispositivo. Se cuenta con numerosas plantillas que simulan distintos tipos de sensores, ya sea de luminosidad, intensidad, detección de humos, etc. Podemos también seleccionar el número de sensores que hay con esa configuración, así como, el *Topic*, *User Name*, *Password*, para el caso de MQTT, entre otras configuraciones.

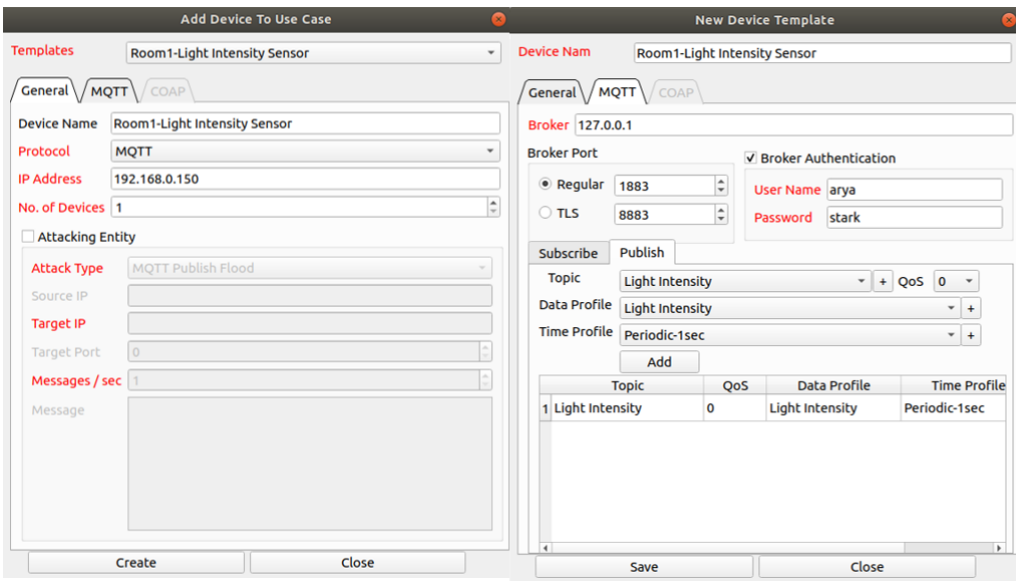

Figura 5.2: Ejemplo de creación de un sensor IoT en IoT-Flock, caso MQTT.

Para el caso del protocolo CoAP, las opciones varían con respecto a MQTT, dadas las diferencias entre ambos protocolos. Como podemos ver en la **Figura 5.3**, se precisa indicar la URL del servidor al que se realizarán las peticiones, así como, indicar el comando que se quiere realizar (*GET, PUT, POST, DELETE*)

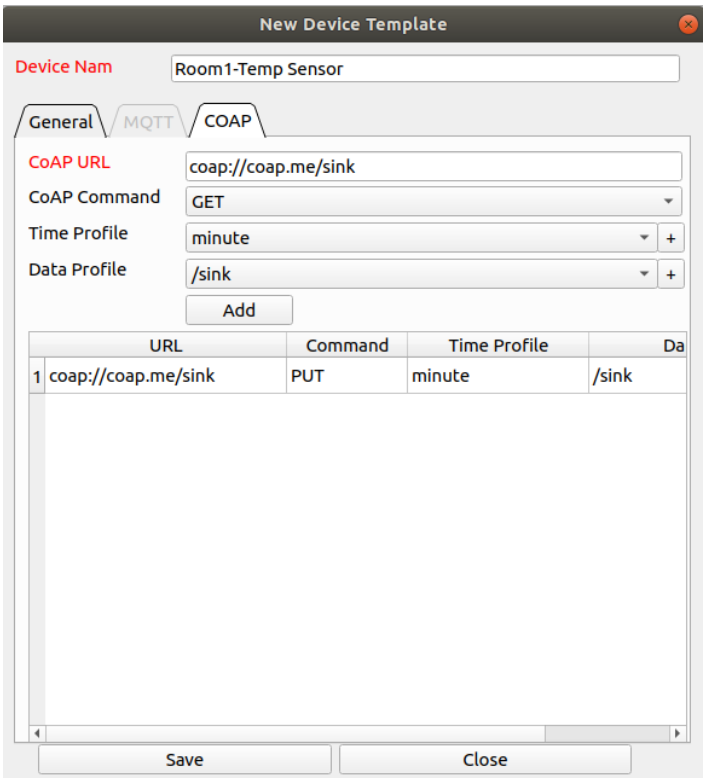

Figura 5.3: Ejemplo de creación de un sensor IoT en IoT-Flock, caso CoAP.

Una vez que se han configurado tantos sensores como se deseen, se necesita exportar esta configuración. Para ello, la herramienta permite guardar los casos de uso en un archivo *XML*. Este archivo es al que se accede mediante el comando de ejecución, el cual se encarga de realizar la conexión y comenzar a transmitir los paquetes deseados. El comando en cuestión es el siguiente:

```
sudo ./IoTFlock-Console nombre_caso_de_uso.xml
```

Con esto, se inicia la simulación de la red y, mediante el uso de Wireshark, se puede comenzar a capturar las trazas que luego se usan para testear el funcionamiento del sistema desarrollado.

Por otro lado, se ha comentado que esta herramienta permite también simular ataques en la red. Uno de los ataques que se han testeado es el ataque por inundación. Estos ataques se caracterizan por el envío de paquetes con grandes cantidades de bytes en cada uno de ellos, estableciendo numerosas solicitudes a los servidores. Esto generará que la ventana TCP de recepción llegue a su límite y a la vez obliga al servidor a contestar a todas las peticiones, pudiendo llegar bloquearlos o ralentizar la red.

5.4. Validación de la carga de datos

Una vez se verificó que la primera parte del desarrollo era correcta, es decir, que se estaba extrayendo de forma adecuada los parámetros que queremos analizar, se da paso a la inserción de estos a la base de datos.

Para poder verificar el correcto funcionamiento del paso de los registros del archivo *CSV* a las bases de datos, se hicieron varias comprobaciones. En el caso de MySQL, tras la ejecución de cada comando se devuelve un código indicando el estado de este. Por ejemplo, el resultado mostrado a continuación es el que se indica tras transferir un archivo con 50000 lineas de registros hacia la base de datos:

```
Query OK, 50000 rows affected, 0 warnings (0.69 sec)
Records: 50000 Deleted: 0 Skipped: 0 Warnings: 0
```

Con esto, podemos ver que los registros están siendo escritos, sin perder ninguno durante el proceso y, además, no se está produciendo ningún tipo de error. También, se puede ver el tiempo que ha tomado la inserción, en el ejemplo mostrado 0.69 segundos. A mayores, se mostraba la tabla construida para verificar, a gran escala, si los registros estaban siendo añadidos de manera adecuada. Para esto, se usa la siguiente consulta, con la que se muestra la tabla generada en su totalidad:

```
SELECT* FROM <nombre de la tabla> (MQTT o CoAP)
```

De cara a la verificación de la inyección de los registros a Elasticsearch, se ha usado en el archivo de configuración de Logstash, la opción *stdout* que, como se mencionó en la **Sección 4.3.2**, imprime por pantalla todos los registros que se están transfiriendo en tiempo real. No obstante, esta verificación únicamente sirve para comprobar, a gran escala, que los parámetros están siendo almacenados de manera correcta y que no hay erratas visibles.

Para indagar un poco más en estos registros de Elasticsearch, se hace uso de la herramienta Kibana. Esta es conocida por ser la primera herramienta que representaba la base de datos de Elasticsearch. Sin embargo, para este desarrollo se ha utilizado el apartado de *Index Management*. Mediante este apartado, se puede editar y conocer el estado de los índices creados en la base de datos, así como verificar el número de registros realizados, los bytes escritos, el tiempo transcurrido, etc.

Con estas verificaciones, se pudo comprobar el correcto funcionamiento de las bases de datos.

5.5. Pruebas de rendimiento

Una vez comprobada el correcto funcionamiento de las bases de datos, nos llega la duda de cual es la más adecuada para el desarrollo que se ha realizado.

Dado que las diferencias en cuanto a posibilidad de uso, en el caso de Grafana son exactamente las mismas, a excepción de la posibilidad de incluir los percentiles en Elasticsearch sin necesidad de realizar ninguna operación adicional, descartamos la funcionalidad que proporcionan estas bases de datos como método de comparación entre ambas.

Una prueba que se ha realizado son los tiempos de carga, tanto de inyección en la base de datos, como de carga de la interfaz de Grafana con los resultados. Para la realización

de estas pruebas se han seguido distintos procedimientos según la base de datos tratada. Para el caso de la comprobación de tiempos en la inyección de los registros en la base de datos MySQL, se han usado los siguientes comandos:

```
Set profiling = 1
Show profiles
```

El primero indica que se quiere almacenar los tiempos de ejecución en una tabla, y el segundo muestra por pantalla dicha tabla.

Para comprobar esta misma medida en la base de datos de Elasticsearch, se vuelve a hacer uso de Kibana y su apartado estadísticas, dentro de *Index Management*. En él, se muestra el tiempo que se ha requerido para inyectar los paquetes en la base de datos. Un ejemplo de ello, lo podemos ver en la *Figura 5.4*

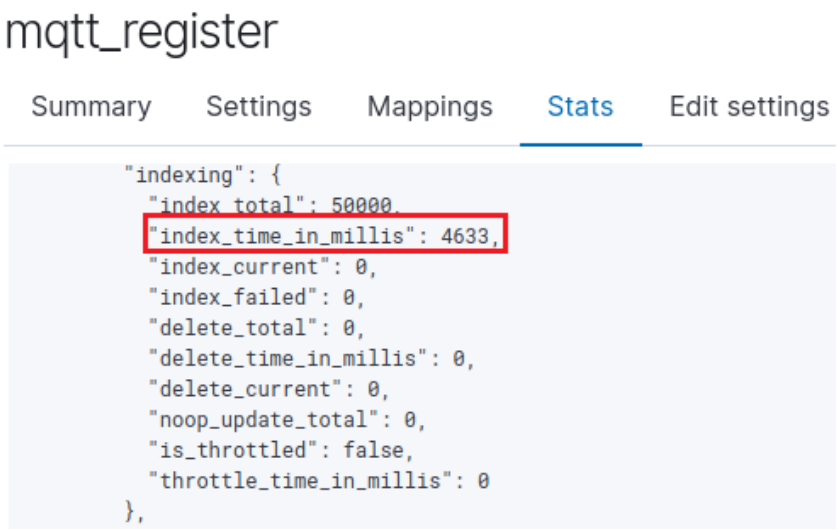

Figura 5.4: Ejemplo de tiempos de carga en la base de datos de Elastisearch de una traza MQTT con 50000 paquetes.

Una vez que se han comprobado los tiempos de inserción de datos en las bases de datos, también se realiza la comprobación de tiempos de acceso a estas, mediante la carga de gráficas en Grafana. Para realizar esta comparativa, se realizaron dos paneles en Grafana con exactamente las mismas representaciones, de manera que se pudiera comprobar cual poseía mejores tiempos de acceso a ellas.

Esta comprobación se realizó con el navegador Firefox. En este, se selecciona la opción de desarrollador web y posteriormente el apartado de red. Con esto, se puede comprobar el tiempo de carga de Grafana ante las peticiones realizadas. Cabe mencionar que estas pruebas se realizaron asegurándose que no hay nada almacenado en la caché y de forma equiparable para ambas bases de datos. Un ejemplo de los resultados obtenidos lo podemos ver en la **Figura 5.5**.

Una vez mencionados ejemplos de las pruebas realizadas, se muestra en la **Figura 5.6** una tabla comparativa en la que se han tomado 6 muestras de cada tipo de prueba para cada una de las dos bases de datos probadas. Luego de realizarlas, se calcula la media y la desviación típica de estos resultados para poder así establecer unas conclusiones. Estas muestras se realizan con una base de datos de 20 minutos, con 50000 paquetes y de MQTT como protocolo.

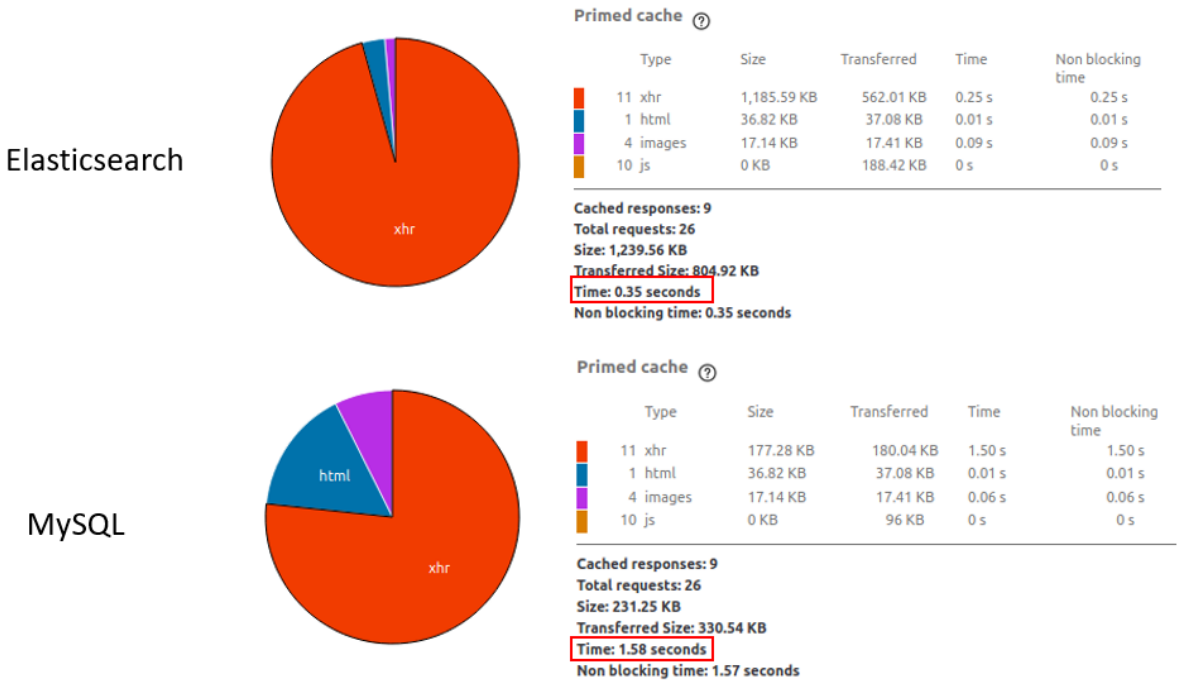

Figura 5.5: Comparación de tiempos de carga, en segundos, de Elasticsearch y MySQL con respecto a la petición por parte de Grafana.

	Medida de carga de grafana	
	Elastic	MySQL
	0,35	1,58
	0,34	1,61
	0,26	1,45
	0,46	1,33
	0,35	2,01
	0,46	1,76
Media	0,370	1,623
Desviación típica	0,077	0,239

	Tiempos de carga bases de datos	
	Elastic	MySQL
	6,389	0,681
	5,745	0,998
	5,666	0,855
	4,633	0,975
	5,452	0,775
	6,140	0,966
Media	5,671	0,875
Desviación típica	0,611	0,127

Figura 5.6: Tabla comparativa de tiempos de carga, en segundos, de Elasticsearch y MySQL.

Podemos ver que, según el proceso que se tome en cuenta, una base de datos sale claramente como favorita con respecto a la otra.

Por un lado, en los tiempo de carga de Grafana, claramente sale beneficiado el uso de Elasticsearch como base de datos, con una media de tiempos de carga de algo más de un cuarto de segundo, a diferencia con MySQL que supera el segundo y medio. Esto era algo que esperábamos, ya que Elasticsearch se caracteriza por la rapidez a la hora de realizar consultas a la base de datos. Por otro lado, en el caso de la carga de las bases de datos, es decir, el tiempo que se tardan en insertar los registros a ellas, MySQL ofrece tiempos muy inferiores, llegando a ser, de media, 6 veces más veloz que Elasticsearch.

En este punto, surge la duda de cual sería la base de datos más adecuada según el caso de uso que estamos analizando. Dado que este trabajo se centra en la capacidad de

monitorización y, por lo tanto, la inmediatez en cuanto a carga de los gráficos se considera primordial, se toma la decisión que Elasticsearch es la más adecuada. Además, dado que la inserción en la base de datos tiene, de media, unos tiempos de cerca de los 6 segundos para trazas de más de 20 minutos, se considera que estos valores son asumibles y se otorga un mayor peso a la capacidad de reacción que se tiene con respecto a las peticiones de Grafana.

5.6. Conclusión

Con este capitulo se concluye el desarrollo del sistema de monitorización para los protocolos MQTT y CoAP en redes IoT.

En él, se ha comentado las diversas estrategias llevadas a cabo para formalizar la validez del desarrollo. Comprobando, al inicio, la correcta extracción de los parámetros de los paquetes, con ayuda de Wireshark y comprobando que se estaban insertando adecuadamente los registros a las bases de datos. También, se ha hablado de las herramientas de generación de tráfico, que han permitido testear el sistema completo. Y, por último, se ha comentado acerca de las pruebas de rendimiento llevadas a cabo, con las que se declara a Elasticsearch como la base de datos a usar en este proyecto.

Una vez finalizadas las validaciones y pruebas se continua con el capítulo de Conclusiones y Trabajo Futuro, en donde se hará un resumen del trabajo realizado y diversas propuestas de mejora.

6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

Durante este Trabajo Fin de Máster se ha desarrollado un sistema de monitorización de redes IoT, más concretamente, de los protocolos MQTT y CoAP. Con este desarrollo, se puede analizar el estado de una red IoT aportando fuentes de información que ayuden a determinar el estado de salud de esta, así como detectar posibles anomalías.

Se han explicado el concepto de IoT, así como, las principales características de los protocolos MQTT y CoAP, ayudando a poner en contexto al lector. También, se han detallado los requisitos funcionales y no funcionales que se han tomado en cuenta a la hora de desarrollar el proyecto.

En cuanto al desarrollo en sí mismo, se ha podido ver el trabajo realizado, de una manera cronológica. Comenzando con el análisis de los paquetes capturados de la red, extrayendo las características que se han considerado relevantes para la monitorización. Seguido de la inserción de los registros, generados tras el análisis, hacia las distintas bases de datos testeadas. Estás últimas, gracias a la realización de pruebas de rendimiento, se ha optado por tomar como la más adecuada, dados los requisitos del proyecto, a Elasticsearch. Por último, se accede a esta base de datos por medio de Grafana, generando tableros donde se muestran los datos en forma de gráficas y tablas.

A su vez, se ha validado el desarrollo, haciendo uso de herramientas como Wireshark apoyado de la generación de tráfico propio. Con esto, se ha podido determinar que el funcionamiento del sistema es el esperado.

Cabe también mencionar, que la realización de este proyecto ha ayudado a mejorar los conocimientos obtenidos en el máster cursado. Gracias a la asignatura de Tecnologías y Servicios de Internet, se ha podido comprender con mayor facilidad las tecnologías y protocolos de IoT. De la misma manera, la asignatura de Planificación de Redes ha permitido tener un mayor conocimiento acerca del tratamiento de datos de medidas de red. También ha sido de ayuda de cara a plantear el proyecto con una capacidad de

escalado futura. También, debemos destacar la asignatura Gestión de Redes, sin la cual la existencia de este proyecto no tendría sentido. Gracias a ella se ha tenido en cuenta la visión que tendría un gestor de red de cara a la administración y mantenimiento de una red IoT, permitiendo perfilar de una manera más adecuada los requisitos necesarios para el sistema de monitorización. Por último, debemos mencionar la asignatura de Proyectos en Ingeniería de Telecomunicación, gracias a la cual se ha seguido una metodología Agile para el desarrollo del proyecto.

Adicionalmente, este proyecto se encuentra íntegramente alojado en el repositorio de Github del siguiente enlace, que se puede acceder escaneando el código QR de la **Figura 6.1**:

`https://github.com/davidsanchesgomez/IoT-Monitor-system.`

Figura 6.1: Código QR con el repositorio Github del proyecto.

6.2. Trabajo futuro

Para una futura continuación de este proyecto, se proponen las siguiente ideas sobre el camino a seguir, de cara a aumentar las funcionalidades del sistema desarrollado:

- **Automatizar todo el proceso:** Mediante un proceso de automatización, que permita realizar todos los procesos de forma mecánica sin intervención del usuario, se podrá permitir un flujo de trabajo más eficiente y facilitará el uso de la herramienta. Pudiendo incluso añadir interfaces de usuario que permitan modificar ciertos parámetros, como puede ser la periodicidad del análisis de la red o incluso insertar múltiples registros de forma simultanea.
- **Añadir captura de tráfico en tiempo real:** Una vez logrado el proceso de automatización, se podría añadir la capacidad de capturar el tráfico en tiempo real, en vez de partir de capturas de tráfico. Mediante el uso de la función `pcap_open_live`, en vez de `pcap_open_offline`, se permite la captura de tráfico en tiempo real, el cual podría ir inyectándose en la base de datos según se vaya capturando.
- **Aumentar el sistema de alarmas ante ataques en red:** La búsqueda de nuevas formas de detectar anomalías debe de ser uno de los principales objetivos de cara a continuar con el proyecto. Una expansión del sistema de alarmas ante otro tipo de ataques supone un gran añadido a la funcionalidad de la herramienta.

- **Profundizar en la generación de gráficos:** mediante el uso de la herramienta por profesionales dentro del sector, se podría obtener información acerca de los datos que consideran más interesantes de cara a una mayor focalización en cuanto a las gráficas y parámetros mostrados.
- **Implementar medidas ante la fragmentación IP:** añadir al analizador de protocolos la capacidad de analizar paquetes que cuenten con fragmentación de IP.
- **Análisis del protocolo MQTT_SN:** realizar un estudio e implementación del protocolo MQTT especializado para sensores inalámbricos, para poder añadirlo al sistema de monitorización.
- **Análisis de los niveles de enlace y red:** aumentar la funcionalidad del analizador de protocolos para poder analizar tráfico con otros tipos de capas de enlace y red. Por ejemplo 6LoWPAN, Zigbee, Bluetooth, etc.
- **Análisis del mercado:** realizar un análisis del mercado en profundidad para comprobar si merece la pena continuar e invertir más recursos en la mejora de la herramienta.

Bibliografía

[1] Suwimon Vongsingthong and Sucha Smanchat. Internet of things: a review of applications and technologies. *Suranaree Journal of Science and Technology*, 21(4):359–374, 2014.

[2] Oracle. What is the internet of things (iot)? `https://www.oracle.com/internet-of-things/what-is-iot/`, 2020.

[3] C. Borman Z. Shelby, K. Hartke. The Constrained Application Protocol (CoAP). `https://tools.ietf.org/html/rfc7252#section-1`, june 2014.

[4] OASIS Standard. Mqtt version 3.1.1. 1, 2014. `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html`.

[5] Mónica Martí, Carlos Garcia-Rubio, and Celeste Campo. Performance evaluation of CoAP and MQTT_SN in an iot environment. *Proceedings*, 31(1), 2019.

[6] Nitin Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE international systems engineering symposium (ISSE)*, pages 1–7. IEEE, 2017.

[7] RedHat. ¿Qué es el internet de las cosas? `https://www.redhat.com/es/topics/internet-of-things/what-is-iot`, 2020.

[8] Vector Software. Embedded computing design. `https://www.embedded-computing.com/`, 2020.

[9] Sanjeev Kaushik Ramani and SS Iyengar. Evolution of sensors leading to smart objects and security issues in iot. In *International Symposium on Sensor Networks, Systems and Security*, pages 125–136. Springer, 2017.

[10] Jasmin Guth, Uwe Breitenbücher, Michael Falkenthal, Paul Fremantle, Oliver Kopp, Frank Leymann, and Lukas Reinfurt. A detailed analysis of iot platform architectures: concepts, similarities, and differences. In *Internet of everything*, pages 81–101. Springer, 2018.

[11] Shohin Aheleroff, Xun Xu, Yuqian Lu, Mauricio Aristizabal, Juan Pablo Velásquez, Benjamin Joa, and Yesid Valencia. Iot-enabled smart appliances under industry 4.0: A case study. *Advanced engineering informatics*, 43:101043, 2020.

[12] Hugh Boyes, Bil Hallaq, Joe Cunningham, and Tim Watson. The industrial internet of things (iiot): An analysis framework. *Computers in industry*, 101:1–12, 2018.

[13] Adrián Mihai Rosu and Fernando Jesús López Colino. Análisis de soluciones inalámbricas para la creación de un sistema de comunicaciones para su uso en dispositivos iot. Final Degree thesis, Universidad Autónoma de Madrid(Escuela Politécnica Superior), 2019.

[14] 20 años de mqtt. `https://www.fundacionctic.org/es/actualidad/20-anos-de-mqtt`, 2019.

[15] Steve. Understanding the MQTT Protocol Packet Structure. `http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/`, enero 2020.

[16] Steve. MQTT Retained Messages Explained. `http://www.steves-internet-guide.com/mqtt-retained-messages-example/`, octubre 2020.

[17] The HiveMQ Team. Client, Broker / Server and Connection Establishment - MQTT Essentials: Part 3. `https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/`, julio 2019.

[18] ntop. nDPI. Open and Extensible LGPLv3 Deep Packet Inspection Library. `https://www.ntop.org/products/deep-packet-inspection/ndpi/`, junio.

[19] Petr Matoušek, Ondřej Ryšavỳ, and Matěj Grégr. Security monitoring of iot communication using flows. In *Proceedings of the 6th Conference on the Engineering of Computer Based Systems*, pages 1–9, 2019.

[20] Leonel Santos, Carlos Rabadão, and Ramiro Gonçalves. Flow monitoring system for iot networks. In *World Conference on Information Systems and Technologies*, pages 420–430. Springer, 2019.

[21] An introduction to cooja. `https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja`, 2019.

[22] Christian Tzolov. CoAP-Shell. `https://github.com/tzolov/coap-shell`, enero 2020.

[23] Syed Ghazanfar, Faisal Hussain, Atiq Ur Rehman, Ubaid U Fayyaz, Farrukh Shahzad, and Ghalib A Shah. Iot-flock: An open-source framework for iot traffic generation. In *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, pages 1–6. IEEE, 2020.

[24] Tetcos. NetSim-Internet Of Things. `https://www.tetcos.com/netsim-iot.html`.

[25] Martin Glinz. On non-functional requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 21–26. IEEE, 2007.

[26] David Sanches Gómez. Desarrollo de un sistema eficiente de análisis del protocolo iec 60870-5-104 para la detección de anomalías en redes scada. Final Degree thesis, Universidad Autónoma de Madrid(Escuela Politécnica Superior), 2019.

[27] Mark Drake. How To Install MySQL on Ubuntu 18.04. `https://www.digitalocean.com/community/tutorials/como-instalar-mysql-en-ubuntu-18-04-es`, abril 2019.

[28] Víctor Cuervo. ¿Qué es Elasticsearch? `http://www.arquitectoit.com/elasticsearch/que-es-elasticsearch/`, febrero 2019.

[29] Christos Tachtatzis, Robert Atkinson, Ethan Bayne, and Xavier Bellekens. MQTT Internet of Things Intrusion Detection Dataset. `https://ieee-dataport.org/open-access/mqtt-iot-ids2020-mqtt-internet-things-intrusion-detection-dataset`, agosto 2020.

[30] ThingzDefense. IoT-Flock. `https://github.com/ThingzDefense/IoT-Flock`, marzo 2020.